Design and Analysis of Secure Exam Protocols

Rosario Giustolisi

Supervisor:
Prof Peter Y.A. Ryan (Université du Luxembourg)

Daily advisor:
Dr Gabriele Lenzini (Université du Luxembourg)

UNIVERSITÉ DU LUXEMBOURG

PhD-FSTC-2015-47

The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 26/10/2015 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Rosario GIUSTOLISI
Born on 16 October 1983 in Giarre (Italy)

Design and Analysis of Secure Exam Protocols

Dissertation defense committee

Dr Peter Y.A. Ryan, dissertation supervisor
Professor, Université du Luxembourg

Dr Gabriele Lenzini, vice-chairman
Université du Luxembourg

Dr Sjouke Mauw, chairman
Professor, Université du Luxembourg

Dr Steve Schneider
Professor, University of Surrey

Dr Luca Viganò
Professor, King's College London

Abstract

Except for the traditional threat that candidates may want to cheat, exams have historically not been seen like a serious security problem. That threat is routinely thwarted by having invigilators ensure that candidates do not misbehave during testing. However, as recent exam scandals confirm, also invigilators and exam authorities may have interest in frauds, hence they may pose security threats as well. Moreover, new security issues arise from the recent use of computers, which can facilitate the exam experience for example by allowing candidates to register from home. Thus, exams must be designed with the care normally devoted to security protocols.

This dissertation studies exam protocol security and provides an in-depth understanding that can be also useful for the study of the security of similar systems, such as public tenders, personnel selections, project reviews, and conference management systems. It introduces an unambiguous terminology that leads to the specification of a taxonomy of various exam types, depending on the level of computer assistance. It then establishes a theoretical framework for the formal analysis of exams. The framework defines several authentication, privacy, and verifiability requirements that modern exams should meet, and enables the security of exam protocols to be studied. Using the framework, we formally analyse traditional, computer-assisted, and Internet-based exam protocols. We find some security issues and propose modifications to partially achieve the desired requirements.

This dissertation also designs three exam protocols that guarantee a wide set of security requirements. It introduces a novel protocol for Internet-based exams to thwart a malicious exam authority with minimal trust assumptions. Then, it proposes secure protocols suitable for both computer-assisted and traditional pen-and-paper exams. A combination of oblivious transfer and visual cryptography schemes allows us to overcome the constraint of face-to-face testing and to remove the need of a trusted third party. Moreover, the protocols ensure accountability as they support the identification of the principal that is responsible for their failure. We evaluate the security of our protocols by a formal analysis in ProVerif.

Finally, this dissertation looks at exams as carried out through a modern browser, Safe Exam Browser (SEB). It was specifically designed to carry out Internet-based exams securely, and we confirm it immune to the security issues of certificate validation. Using UML and CSP, we advance a formal analysis of its requirements that are not only logically conditioned on the technology but also on user actions. By extending this analysis onto other browsers, we state general best-practice recommendations to browser vendors.

Acknowledgements

Pursuing a Ph.D. is a journey that involves many people. Here I want to acknowledge the people without whom this journey would not be The Journey.

First, I would like to express my gratitude to Gabriele Lenzini for his continuous supervision. He has devoted much of his time to help and support me in this journey. He has patiently taught me how to evaluate my work and how clearly formulate the results. I am thankful to Peter Ryan for accepting me in his research group and for his valuable advice during our conversations. I am fascinated of his rare ability to quickly grasp the essence of any mathematical problem. I am particular grateful to Giampaolo Bella for his precious support and advice. He has started me on research and is always up to taught me how to develop my research skills further. Moreover, he is a great motivator.

I would like to thank my Ph.D. examiners Sjouke Mauw, Steve Schneider, and Luca Viganò for their interest and valuable expertise in giving some advice useful for my dissertation. I am thankful to the Doctoral School of Computer Science and Computer Engineering for the financial support provided to attend conferences and summer schools.

I am particular grateful to Jannik Dreier, Ali Kassem, and Pascal Lafourcade for the fruitful collaboration on the formal specification of exam requirements. I would like to thank Andrea Huszti for the interesting discussions about the security of exam protocols and efforts on how improve them. I also enjoyed discussions with people I met in scientific conferences and summer schools. In particular, the annual Workshop on Security Frameworks has been a great source of feedback and ideas, and I benefited from discussions with people I met there (Giampaolo Bella, Denis Butin, Gianpiero Costantino, Gabriele Lenzini, Giuseppe Patanè, Salvatore Riccobene, ...).

I had great time with my fellow Ph.D. students (Arash, Massimo, Afonso, Jean-Louis, Miguel, Marjan, Dayana, Masoud, Jun) at ApSIA, and never felt too far from home thanks to my Italian office neighbours (Claudio and Vincenzo). I was privileged to live at the Résidence des Dominicaines and to have a lot of friends living there.

I would like to thank my family: Daniela and Leo for taking care of everything during my stay abroad; my parents Franco e Pina for their unconditional love and support they have been giving to me, and for have allowed me to be the person I am today; my girlfriend Silvia for standing by me, for having patiently shared successes and failures, and for always believing in me.

Finally, I would like to thank you for reading this dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Oxford English Dictionary defines an *exam* as *the process of testing the knowledge or ability of pupils, or of candidates for office, degrees, etc.* Although the definition of exam is pivoted on individuals, exams assume a key role in fostering meritocracy in modern societies. On the individual side, exams are important as they help people understand their skills and knowledge in a particular subject. On the societal side, exams are indispensable to select most appropriate people through an objective evaluation.

The use of exams is widespread, with various examples derivable from the education sector (e.g., admissions, coursework, and final qualifications) as well as from the work sector (e.g., recruitment, progression, and professional qualifications). France is one of the countries that extensively use exams, some of which are very competitive: the "Concours Général" is the most prestigious academic exam, which is taken by about 15,000 French students, and typically bears a success rate of less than 2% [Min14]; the exam to enter medicine studies attracts every year more than 50,000 candidates with a success rate of less than 15% [Fig14]. Exam meritocracy is one of the guiding principles in Singapore and is deemed to have contributed to the rapid growth of the country [Gop07]. High populated countries, such as China and India, resort on tough exams for the recruitment of various administrative officers. In particular, China has an old tradition in administering civil service exams, which dates back to the Han dynasty (206 BC - AD 220). It is worth noting that the idea to promote people based on exams came to West only in the 17th century, when the British Empire began to hire employees using competitive exams to eliminate favouritism and corruption [KER09].

Exams fulfil the goal of testing knowledge and abilities of candidates only in absence of misbehaviour of the involved parties, and normally employ different methods to face possible threats. For example, threats ascribed to candidate cheating are normally mitigated by invigilation and anti-plagiarism methods. Moreover, an exam requires people to follow many procedures to make sure that as few things as possible go wrong. For example, to ensure that only eligible candidates attend the exam, the invigilator checks their identity documents before allowing them to take the exam. To guarantee fair marking, each candidate may use a special number that replaces her name on the test so that the examiner can mark the test while ignoring its author. The author of a test is revealed only after the examiner marked the tests.

The procedures outlined above help people's confidence that everything works as expected, provided that people can observe the procedures or trust the exam system. The latter means that people have to trust that the candidate's ID is properly checked, that special numbers are correctly employed, and that authorities do not reveal the author of a test before the examiner marks it. People have to trust authorities.

However, authorities and examiners may be corrupted, and so they may commit misconducts that are hard to eradicate. In the scandal known as Atlanta Cheating [Cop13], about 35 people among school administrators, educators, and superintendents manipulated ranks and scores with the goal of gaining more school governmental funds. In early 2014, the BBC revealed a fraud on the UK visa system, in which the invigilator read out all the correct answers during the exam of English proficiency [Wat14]. More recently, a medical school admission exam scandal in India has turned into thousands of arrests [New15]. The police revealed that candidates hired impersonators to take the written exam, and examiners gave higher marks to colluded candidates. The U.S. Navy disclosed cheating on the written exam that concerns the use of nuclear reactors that power carriers and subs [Lip14]. It was found that questions and answers were illegally taken from a Navy computer since 2007 [Pre14].

In fact, computers have been increasingly introduced in the main procedures of exam systems. They can assist generation of questions or automatic marking procedures; also, they can be employed for remote registration, remote notification, and even remote testing, in which candidates can take the exam from home. For example, the most popular Massive Online Open Courses (MOOC), the education platforms that offer courses of study over the Internet, allow remote testing [Cou15, Cou12]. Such exams provide a formal recognition of the candidates' achievements, and some universities already consider MOOC exams eligible for university credits [Lew13].

The use of computers simplifies certain tasks occurring during an exam, but does not necessarily make the exam more secure. For example, the registration of a candidate for the exam and the notification of the mark via the Internet should be at least as secure as they would be face-to-face. Hence, we shall unfold the argument that an exam must be designed and analysed as carefully as security protocols normally are.

This dissertation draws its main motivation from the observation that exams raise more challenging security and privacy issues than one may think. This is due to at least two main reasons. One is that threats may come from any of the roles playing in an exam. In particular, candidates and authorities may be corrupted to various extents. Exams then begin to look more balanced in terms of threats or benefits their participants pose or seek. The other reason is that there is no clear understanding of what the relevant security requirements for exams are. We observe that the growth in the use of exam protocols has neither been followed nor preceded by a rigorous definition and analysis of their security. Such absence may lead to the design and the practical adoption of insecure exam protocols, and makes people less confident of exam trustworthiness.

These concerns are relevant also for other domains, such as voting. Significant parts of the related work in this dissertation highlight similarities and differences between exams and voting. Intuitively, both domains share similar security and privacy requirements. For example, only the answers originated by eligible candidates should be marked in an exam. Similarly, only ballots cast by

eligible voters should be recorded in a voting system. However, a closer inspection reveals a number of different security concerns between the two domains. For example, the link between an answer and its author should be preserved through the phases of an exam. Conversely, unlinkability between voter and vote is a desired property of a voting system. In a nutshell, a fundamental difference is that while fair elections aim to bring *democracy*, fair exams aim to bring *meritocracy* to societies. We observe that democracy is not meritocracy: in democracy, the selection of candidates is based on (people's) choices; in meritocracy, the selection of candidates is based on (the examiner's) assessment of their merit. Thus, understanding similarities and differences between exams and voting becomes an additional motivation of this manuscript.

1.1 Aims and Objectives

This work aims to study the relevant security requirements for exam protocols and to design and analyse exam protocols that meet the stated security requirements. We intend to achieve those aims through four objectives.

1. **To identify the relevant security requirements for exam protocols.** This is a fundamental objective as it provides the basics for further research and determines the meaning of *secure* exam protocol. It requires the specification of a coherent terminology for exams including their phases and threat model. The desired outcome consists of a set of authentication, privacy, and verifiability requirements.

2. **To develop a formal framework for the specification of security requirements and the analysis of exam protocols.** This is a crucial objective that provides a rigorous and formal description of the security requirements for exams. It requires choosing a specific formalism in which the security requirements identified in objective 1 can be expressed. The desired outcome is to achieve a flexible framework that is suitable for the modelling and analysis of exams.

3. **To design new secure protocols for different types of exams.** This objective consists in proposing novel exam protocols that meet the security requirements according to the restrictions of the different exam types, which depend on the level of computer assistance and span from traditional to Internet-based exams. It requires combining secure cryptographic schemes to guarantee the often contrasting requirements. The desired outcome is a number of protocols that provide the same level of security though they belong to different exam types.

4. **To explore novel security aspects of the critical components of exam protocols.** This objective is to expand the formal analysis of exams by considering also the user. In particular, it concerns the analysis of one of the components that interacts with the user most. As we shall see later, this component is the browser. This objective requires choosing a suitable approach that includes the user in the formulation of security requirements and in the analysis. The desired outcome is to understand how user's choices may influence the security of exam protocols.

Objective 1	A clear description of the general building blocks of all exams, and the definition of the security requirements
Objective 2	Two formal frameworks for the security analysis of exam protocols
Objective 3	Three new exam protocols that guarantee several security requirements
Objective 4	A socio-technical formal analysis of browsers for Internet-based exams

Figure 1.1: Overview of the contributions

1.2 Contributions

This dissertation addresses the four objectives outlined in the previous section. It advances the state of the art in the design and analysis of secure exam protocols with at least four original contributions as reported in Figure 1.1.

The first contribution is a clear specification of the general building blocks of all exams, in terms of tasks, roles, phases, and threats. This paves the way for the definition of the security requirements for exams and facilitates the description of exam protocols.

The second contribution consists of two formal frameworks for the security analysis of exam protocols. The frameworks enable the study of five authentication, five privacy, and eleven verifiability requirements for exam protocols, and support the specification of additional requirements. One framework formalises authentication and privacy requirements in the applied π-calculus [AF01], while the other specifies verifiability requirements in a more abstract model . Both frameworks are validated in ProVerif [Bla01] on traditional, computer-assisted, and Internet-based exam protocols. We find that some protocols are flawed and propose modifications on their designs.

The third contribution consists of three new exam protocols that guarantee a set of security requirements. The first protocol is for Internet-based exams and meets authentication, privacy, and verifiability requirements with minimal reliance on trusted parties. It distributes the trust across the servers that compose an *exponentiation mixnet*. The second protocol is for computer-assisted exams with face-to-face testing, and meets the requirements by means of lightweight participation of a trusted third party (TTP). It exploits the use of signatures and visual cryptography to ensure authentication and privacy in the presence of corrupted authorities and candidates. The last protocol eliminates the need of the TTP by combining oblivious transfer and visual cryptography schemes. It still ensures the same security requirements as the previous protocol's and provides accountability without relying on a TTP. The proposed exam protocols are formally analysed in ProVerif.

The last contribution is the formal analysis of six modern browsers for

Internet-based exams. This analysis tackles yet another aspect of the problem of secure exams as it concerns the human understanding of remote authentication, and in particular of certificate validation in browsers. Browsers are critical components of an exam as they are the main application users normally interact with. The list of analysed browsers includes Firefox, Chrome, Safari, Internet Explorer, Opera Mini, and Secure Exam Browser (SEB), a kiosk browser that enforces the security of remote testing in Internet-based exams. Using UML Activity Diagrams and the CSP process algebra [Hoa78], we provide a systematic approach to the security analysis of certificate validation in different scenarios, all considering user interactions. Parts of this contribution are the Linear Temporal Logic specification of five novel socio-technical requirements, each binding elements like TLS session, certificates and user choices. We find that each browser implements certificate validation differently, and some of them pose more security risks to users than others do. We propose four recommendations to improve the security of browsers' certificate validations.

The results of this research offer the basis for the design and analysis of secure protocols for traditional, computer-assisted, and Internet-based exams.

1.3 Outline

This dissertation is structured in eight chapters. Most of the contents of the dissertation have been published as joint work with different co-authors in conference papers or submitted to journal articles. In the following, we outline the contents of each chapter.

Chapter 2: Terminology. This chapter introduces the basics of an exam. It begins with the description of the tasks that occur during an exam. In particular, it observes that levels of detail and abstraction of an exam specification constraint the number of tasks. The chapter continues discussing the possible roles of an exam, possibly played by one or more principals. It identifies the typical phases of an exam and the basic threats coming from the main exam roles. The chapter concludes with a taxonomy that classifies exams by types and categories.

The contents of this chapter have not been published. However, a paper containing an earlier version of the chapter was accepted for publication in a conference but not included in the proceedings.

Chapter 3: Formalising Authentication and Privacy. This chapter contains the formal definitions of authentication and privacy requirements for exams. It discusses different techniques to model security requirements and presents tools for the automatic analysis of security protocols. Then, it describes the applied π-calculus, on which we rely to build the framework. The framework consists of the formal model of an exam, five authentication, and five privacy requirements. It is validated via the ProVerif analysis of the Huszti-Pethő exam protocol. The chapter continues with the results of the analysis and concludes with a proposal that enhances the security of the Huszti-Pethő protocol.

Most of the contents of this chapter are based on different publications. In particular, the informal definitions of authentication and privacy requirements are based on a joint work with Giampaolo Bella and Gabriele

Lenzini [BGL13c]. The formal framework and the analysis of the Huszti-Pethő protocol have been co-authored with Jannik Dreier, Ali Kassem, Pascal Lafourcade, Gabriele Lenzini, and Peter Y.A. Ryan [DGK+14b]. An extended version of this work that considers the fixes for the Huszti-Pethő protocol has been submitted to a journal. The formalisation of the authentication requirement of Candidate Authorisation and its verification on the different protocols discussed in this dissertation are unpublished work.

Chapter 4: Formalising Verifiability. This chapter proposes an abstract framework in which an exam protocol and its verifiability requirements are formalised via a set-theoretic approach. In this chapter, we distinguish the notions of *individual verifiability* as verifiability from the point of view of the candidate, and *universal verifiability* as verifiability from the point of view of an external auditor. We propose six individual and five universal verifiability requirements.

This chapter is based on joint work with Jannik Dreier, Ali Kassem, Pascal Lafourcade, and Gabriele Lenzini [DGK+15].

Chapter 5: The Remark! Internet-based Exam Protocol. This chapter details *Remark!*, a new protocol for Internet-based exams. It discusses the cryptographic building blocks on which Remark! is based, with a particular focus on the exponentiation mixnet. The chapter continues with the description of the protocol and the formal analysis in ProVerif of authentication, privacy, and verifiability requirements. Notably, it discusses how to map the abstract definitions of verifiability in ProVerif. The chapter concludes with some security considerations of Remark!.

The paper that proposes Remark! has been co-authored with Gabriele Lenzini and Peter Y.A. Ryan [GLR14]. The formal analysis of Remark! is based on the papers that appears in [DGK+14b] and [DGK+15]. The manual induction proofs that support the formal analysis of universal verifiability in Remark! are based on [DGK+14a].

Chapter 6: Computer-assisted Exam Protocols. This chapter focuses on *WATA*, a family of computer-assisted exams each employing some level of computer assistance though keeping face-to-face testing. This chapter first introduces the protocol versions of the existing WATA II and III software and reviews their security. Then, it details *WATA IV*, a novel exam protocol that meets more security requirements than the previous ones with less reliance on a TTP. WATA IV is then redesigned to meet the same security requirements without the need of any TTP. The chapter presents a detailed description of the enhanced version and a formal analysis in ProVerif, including the formalisation of an accountability requirement (Dispute Resolution). It concludes with a brief review of the protocols seen throughout the chapter.

This chapter is based on joint work with Giampaolo Bella, Gabriele Lenzini and Peter Y.A. Ryan. The protocol versions of the WATA software and their informal analyses are unpublished work. WATA IV has been published in a conference paper [BGL14]. The enhanced version and its formal analysis are based on [BGLR15].

Chapter 7: Formal Analysis of Certificate Validation in SEB and Modern Browsers. This chapter considers exams taking place via modern browsers and focuses on the socio-technical analysis of certificate validation. It clarifies the basics of certificate validation and highlights the user involvement in the process. Using UML Activity Diagrams, it describes the certificate validation of six different browsers, considering both classic and private browsing modes. This chapter then introduces five socio-technical requirements in Linear Temporal Logic and outlines the translation of the UML diagrams to CSP# [SLDP09]. It discusses the output of a model checker analysis, and concludes with four recommendations to browser vendors.

This chapter is based on joint work with Giampaolo Bella and Gabriele Lenzini. A preliminary version appears in [BGL13b], while the full treatment is based on [BGL13a]. The analysis of SEB, Safari, private browsing modes, and the interleaving with classic browsing are based on a version submitted to a journal.

Chapter 8: Conclusions. This chapter discusses the research presented throughout the dissertation, outlines future work, and concludes the presentation.

Chapter 2

Terminology

This chapter introduces the reader to the general building blocks of all exams. In consequence, describing a specific exam becomes easier, at the sole price of further expanding or specifying these general concepts. We view an exam as a security protocol that involves various tasks defining roles played by various principals through various phases. Hence, *exam* or *exam protocol* are used interchangeably. With a security take, an exam is expected to withstand a threat model meeting a number of security requirements.

Outline of the chapter. Section 2.1 discusses the levels of detail and abstraction to characterise tasks. Section 2.2 introduces possible roles for exams. Section 2.3 outlines the principals that play the exam roles. Section 2.4 identifies the typical phases of an exam. Section 2.5 details the potential security threats associated with the exam roles. Section 2.6 classifies exams by type and category, and concludes the chapter.

2.1 Tasks

A number of *tasks* may occur during an exam, such as generating the set of questions, building the tests and marking them. We observe that the number of tasks that can be identified may change over two possible dimensions.

One is the required *level of detail* for the specification of the exam protocol, establishing whether a task should be explicitly mentioned or not. For example, classical security protocols often prescribed the task of *using* a nonce in a message, yet omitting the task of *fetching* it. Experience teaches us that a specification should make very clear (in)security assumptions about the protocol environment, otherwise the analysis may yield debatable findings. In this vein, Needham stated that the public-key Needham-Schröder protocol considered the attacker as an outsider but never made this explicit [Nee02], a threat model that would remove the opportunity for Lowe's attack.

Another dimension is the *level of abstraction* for the specification, establishing whether a task should be expanded into sub-tasks or not. For example, the task of fetching a nonce may be expanded into accessing a random number generator, running it, and receiving its output securely, and these may be further expanded in turn.

From the security analysis standpoint, the levels of detail and of abstraction must be chosen with care in order to limit the necessary assumptions to realism.

2.2 Roles

A *role* is a set of principals who perform a specific set, possibly of cardinality one, of tasks. During exams, an obvious role is the *candidate* role, of taking the exam to get a mark that may accord the candidate a goal — such as obtaining a qualification, passing a periodical academic assessment, or being selected through a public examination. Of course, the candidate role could be specified, if needed, at a lower level of abstraction, and examples can be derived from actual protocol specifications. Other possible roles, also called *authority roles*, are as follows.

- The *registrar* role, of checking the eligibility of candidates who wish to take an exam, and of populating a list of registered candidates accordingly.
- The *question committee* role, of building the tests and passing them to invigilators, and of forming the test answers in case of multiple-choice tests and passing them to the examiners.
- The *moderator* role, of liaising with the question committee to independently ensure that the tests conform to pre-existing quality standards, such as readability and appropriateness.
- The *invigilator* role, of distributing tests to candidates, of checking candidates' identities, of following candidates while they take their test preventing them from misbehaving.
- The *collector* role, of collecting the tests from the candidates at the end of the exam time, and distributing the test answers to the examiners.
- The *examiner* role, of reading the test answers and of producing adequate marks for them.
- The *recorder* role, of keeping records of what candidates received what marks at the exam.
- The *notifier* role, of informing the candidates of the marks that their respective tests received, and of storing this information with some recorders.
- The *observer* role, of watching everything.

It can be seen that each authority role clearly indicates its set of tasks, demonstrating the levels of detail and of abstraction that we advocate. If an exam features an additional task, then this could either extend an existing role or form a new role. Also, in order to meet the security requirements, an exam may allow two or more roles to merge into one, or may prescribe splitting a role into two or more. For example, the role of question committee can be split into exam convener, question setter, and question reviewer, as practised in some universities. Typically, candidate and authority roles cannot be merged (and still meet the security requirements), except with exams such as MOOC [YP13], where homeworks are peer-reviewed, namely candidates mark each other.

2.3 Principals

Exams see the participation of a number of *principals*, each playing one or more of the roles defined above. Principals may change depending on the specific exam, and various examples can be made.

At university, the candidate role is played by students, while the roles of invigilator, collector and examiner are sometimes played by a single lecturer. At an extreme, there are only two principals, with a student as a candidate and a lecturer playing all authority roles. Today, ProctorU [Inc15] invigilates via webcams the candidates who take exams from home. At public examinations, the Police could take the invigilator role.

Principals are not necessarily humans. They may as well be pieces of software playing various roles, such as invigilator, by filming candidates during testing, or examiner, by marking multiple-choice tests mechanically.

Irrespectively or whether they are human or not, it must be assumed that principals may act somewhat maliciously. We anticipate that such assumptions are fundamental when considering corrupted principals in the formal definition of our security requirements. In fact, they will define a threat model, as we shall see below.

2.4 Phases

We identify the four main phases that typically take place sequentially during an exam. They will be further detailed and specified by actual exams in the sequel of this work.

- At *preparation*, certain authorities, typically registrars, file a new exam, and check the standard eligibility criteria, such as correct payment of fees and adequate previous qualifications, of candidates who wish to take the exam. Only those who satisfy the criteria get successfully registered, and the authorities ultimately produce a list of candidates registered for the exam. Similarly, the authorities might produce a list of eligible examiners. Most importantly, this phase includes the preparation of tests and all the relevant material for the subsequent phase. For instance, creation of questions, printing of tests, and generation of pseudonyms to anoymise tests are tasks accomplished in this phase.

- At *testing*, each registered candidate gets a test containing a number of questions, which were previously built by authorities, normally question committee and moderators. Other authorities, typically invigilators, watch candidates through this phase. Each candidate answers their test, and may have to complete it with their personal details. The candidate then submits their test answers to an authority, such as an invigilator.

- At *marking*, the test answers of all candidates reach the examiner authority for evaluation. More precisely, the authority reads the test answers and evaluates their adherence to the required knowledge, then forming a mark, chosen on a given scale, for each test. Some real-world example scales are: pass/fail, A to E, 60% to 100% and 18 to 30. With multiple-choice tests, the examiner authority could be a computer program.

- At *notification*, an authority, commonly a notifier, gives each candidate the mark for the test answers the candidate submitted; either beforehand or afterwards, the notifier also stores this information with a recorder, commonly a server equipped with a DBMS.

Any subset of phases may either take place *on site*, with candidates meeting the authorities face-to-face, or *off site* otherwise. Regulated by the specific application requirements, these features will shape up the exam experience.

Moreover, any subset of these phases may take place either traditionally, namely by pen and paper, or on computer. For example, we observed above that marking can be easily computer assisted with multiple-choice tests; similarly, notification could take place via dedicated workstations installed in the exam site.

The specification of the exam phases clarifies the terminology that is used coherently throughout this dissertation, but it may be useful to the reader if we point out that various synonyms are used in the literature. For example, "registration" or "setup" may refer to the preparation phase; "examination" is often taken to indicate the testing phase; "evaluation" or "grading" may indicate the marking phase; "exam", "examination", or "assessment" may sometimes even refer to the full sequence of phases.

2.5 Threats

A number of threats could be envisaged against exams, and some basic threats are enumerated here.

Threats may derive from each task. For example, even the task of printing may invite the principal who performs it to alter the printout or not print at all. We assume each principal to be rational in the sense that the principal does not misbehave unless there is a clear benefit for them or, in case of *collusion*, for another principal.

We conveniently define threats on a per-role basis. Therefore, augmenting a role with additional tasks would require extending the role-specific threats; adding new roles formed of new tasks would require extending the threats over the role; merging roles would yield the union of the threats of the original roles; splitting a role would partition its threats enabling each formed role to pose the threats deriving from its tasks.

We define some basic threats coming from the preeminent roles used in the sequel of this dissertation: the candidate, the authority, and the observer roles. A specific protocol shall customise the list of the threats according its roles.

The threat model is the standard Dolev-Yao [DY83] over the roles (or their portions) that are impersonated by computer programs. Additionally, the roles (or their portions) that are impersonated by humans pose the threats detailed below.

The corrupted candidate performs any tasks in order to:

- register for an exam without being eligible;
- register on behalf of someone else;
- answer their test with knowledge obtained by cheating;

- get a higher mark than what the examiner assigns to their test.

These threats are significant. Eligibility criteria may be stringent, such as payment of fees and restricted participation to certain exam dates, hence the interest in misbehaviour. Candidates' attempts at cheating, for example by consulting books, and at getting a better mark than theirs are well known. In particular, the latter may see a candidate send someone else, more knowledgeable than them, to sit for the exam on the candidate's behalf, or see a candidate swap their mark with that of another candidate known to be very knowledgeable. Thus candidates may collude each other to achieve their goal. It can be anticipated that these threats demand effective authorisation, authentication, invigilation and marking procedures.

The corrupted authority performs any tasks in order to:

- assign an unfair mark to a specific candidate, namely to over-mark or under-mark her, or assign no mark at all.

This is the fundamental threat authorities may pose to candidates. This threat may hinder students' spontaneity during University lectures. This threat is well known in academic conferences, and is partially addressed with blind reviews. More seriously it has also brought corruption into public competitions. Arguably, it calls for anonymous marking, verifiability, and accountability requirements.

The observer performs any tasks in order to:

- gather any private information.

Observers may be allowed to watch parts of the exam, typically the candidates while they take their tests, to raise public acceptance of the regularity of procedures. They may, however, have malicious intentions and seek out any form of private information such as candidates' questions and marks. If the exam is somewhat assisted by computers or by the Internet, then this threat becomes a digital one.

2.6 Taxonomy

Having seen the phases of an exam, we can specify the dictionary definition of the word "exam" conveniently for our purposes.

Definition 1 (Exam) *An* exam *is a formal test taken to show candidate's knowledge of a subject. It comprises the four sequential phases* preparation*,* testing*,* marking *and* notification*.*

Definition 1 only specifies the main functional requirement of an exam, that candidates take the exam. It purposely omits additional functional requirements that may depend on the application scenario, such as that candidates be allowed to register from home.

2.6.1 Exam Types

Exams can be classified in various types according to the following definitions.

Definition 2 (Computer-assisted exam) *A* computer-assisted exam *is an exam such that at least one of its phases receives some level of assistance from computers or Information Technology.*

At first glance, Definition 2 appears to be too wide to the extent that every exam is a computer-assisted exam. For example, an exam that only requires computers to edit the questions could be classified as computer-assisted. However, as observed in Section 2.1, the levels of detail and abstraction for the specification of the protocol should be considered to find the correct classification of the exam. Thus, if the way the questions are edited is not explicitly mentioned in the description of the protocol, the exam should not be classified as computer-assisted. A qualifying example of a computer-assisted exam is to allow candidates to register from home, but then continuing traditionally, namely fully on paper and without the use of computers.

Definition 3 (Traditional exam or non-computer-assisted exam) *A* traditional exam *is an exam such that none of its phases receives any level of assistance from computers or Information Technology in general. A traditional exam is also said a* non-computer-assisted exam *to indicate the absence of computer assistance.*

Definitions 2 and 3 insist that exams can be partitioned between computer-assisted and traditional exams depending on computer assistance. Having seen the main partition within exams, various types of e-exams can be defined. For brevity, it is useful to refer to the acronyms in Table 2.1.

NCA	non-computer-assisted
CA	computer-assisted
CB	computer-based
IA	Internet-assisted
IB	Internet-based

Table 2.1: Acronyms for exam types

Definition 4 (Computer-based exam) *A* computer-based exam *is an exam whose testing phase takes place fully on computer.*

We decide to pivot Definition 2 around the testing phase because an exam is often somewhat simplistically understood as that phase alone in practice. The definition insists that the testing phase of a CB exam takes place fully on computer, ruling out exams where questions are given orally or are written on a board, which would only be CA exams. Clearly, a CB exam also is a CA exam but not vice versa.

Definition 5 (Internet-assisted exam) *An* Internet-assisted exam *is an exam such that at least one of its phases receives some level of assistance from the Internet.*

It is logical that Definition 2 and Definition 5 have the same structure, and similar considerations about levels of detail and abstraction apply here. Definition 5 requires some use of the Internet in some phases. For example, an

$\mathcal{NCA}$	set of all NCA exams
$\mathcal{CA}$	set of all CA exams
$\mathcal{CB}$	set of all CB exams
$\mathcal{IA}$	set of all IA exams
$\mathcal{IB}$	set of all IB exams

Table 2.2: Sets of exam types

exam that only relies on the Internet to notify the candidates of their marks would be an IA exam. Clearly, an IA exam also is a CA exam. An IA exam may also be, but not necessarily, a CB exam. From a set-theory standpoint, if $\mathcal{CB}$ is the set of all CB exams, and $\mathcal{IA}$ is the set of all IA exams, it follows that $\mathcal{CB}$ and $\mathcal{IA}$ intersect but do not coincide.

Definition 6 (Internet-based exam) *An* Internet-based exam *is an exam whose testing phase takes place fully over the Internet.*

The formulation of Definition 6 closely maps one of Definition 4. Clearly, an IB exam is also an IA exam; an IB exam must also be a CB exam because the testing phase could not happen fully over the Internet without happening fully over some computer (or similar devices such as smartphones).

Also, Definition 5 and Definition 6 purposely omit the specification of the venue where the exam phases happen, whether locally, at the hosting institution's premises, or remotely from the candidate's place. For example, even an IB exam could happen locally.

Having defined the various types of exams, the main building blocks for our taxonomy of exams are available. Still, for each exam type, it is useful to define the set of all exams of that type, as Table 2.2 does in a self-explaining form. Now, the exam taxonomy can be introduced using a set-theory notation; it is in Figure 2.1, and demonstrates the relations discussed above between the various exam types.

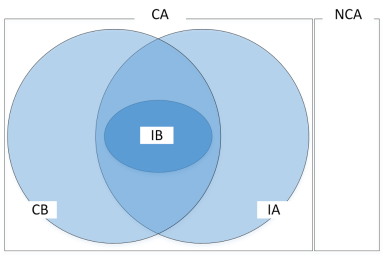

Figure 2.1: A set-theory representation of exams

As anticipated above, all roles span uniformly across the entire taxonomy because principals can either be human or not. Other objects may have to be re-interpreted depending on the exam type. For example, a test consists of some paper (showing questions and, later, answers) for all exams in $\mathcal{NCA}$, of some file(s) for all exams in $\mathcal{CB}$, of either some paper or some file(s) for all exams in $\mathcal{CA} \setminus \mathcal{CB}$.

The taxonomy in Figure 2.1 seems to bear potential to capture more general scenarios than just exams, and precisely those commonly addressed as *collaborative working environment* [FJ95]. However, one would need to map the exam phases identified above to the main phases of other scenarios, and this exceeds the aims of the present research.

2.6.2 Exam Categories

Exams can be categorised as *written exams*, *oral exams*, or *performance exams*, as outlined below. Written exams are usually preferred when the number of candidates is high. Oral exams are considered the most effective practice to assess knowledge, but examiners tend to mark candidates less objectively compared to written exams. Performance exams are impractical when many candidates need to be evaluated.

Our taxonomy was built to reflect written exams, where testing takes place in writing, namely with questions and answers given using various combinations of reading and (hand- or type-)writing on paper, boards, screens, etc. However, our taxonomy can be stretched out to reflect also oral and performance exams as explained in the following, although this exceeds the scope of this dissertation.

Oral exams see questions and answers given orally, either synchronously by interviews or asynchronously by some proxying. Synchronous exams can be represented by our taxonomy, as one can easily realise by looking at Figure 2.1 with synchronous oral exams in mind. For example, a synchronous oral exam in $\mathcal{IB}$ may have the testing phase via videoconferencing; even one in the set $\mathcal{CB} \setminus \mathcal{IA}$ makes perfect sense if recording the interview is required.

Also asynchronous oral exams comply with our taxonomy. Through the use of techniques of speech processing or audio sampling, asynchronous oral exams could have the testing phase via voice chats. Also in this case exams in $\mathcal{CB} \setminus \mathcal{IA}$ are valuable, by yielding the full chat history of the testing. Envisaging an asynchronous oral exam that is NCA requires admitting the role of proxy to be played by a human without any computer assistance. If this is deemed impractical or implausible, then the taxonomy could be easily pruned of the NCA part, only for the category of asynchronous oral exams.

Performance exams differ from oral exams because the former require a candidate to actually perform an activity, rather than answer questions orally. The same techniques envisaged for synchronous and asynchronous oral exams (e.g., videoconferencing, audio sampling, speech processing) hold for performance exams as well as the considerations that regard the exam types.

There may also be *hybrid exams*, for instance where testing combines written and oral means: a lecturer could dictate the questions for candidates to answer in writing. Although it might be less popular than written or oral exams, we argue that this category too could be represented with our taxonomy, perhaps with minor adjustments.

Chapter 3

Formalising Authentication and Privacy

Security protocols are distributed algorithms that use cryptography to achieve some security goals. The design of such protocols can be a difficult task that might lead to serious security flaws. The literature is full of security protocols and standards that have been demonstrated to be bugged [CJS+07, HM05, BGW01], a trend that is unlikely to change [ACC+08, AMRR14]. Formal approaches for the analysis of security protocol have been successfully used to discover security flaws, such as the famous one in the Needham-Schroeder Public Key protocol [Low96], and eventually helped in fixing the protocols with some level of guarantee. Experience shows that formal approaches are important also in the design phase of protocols as they force designers to have a deep understanding of their models and what they aim to achieve. With this belief, we intend to formulate a formal framework for the design and analysis of secure exam protocols.

Similar to other systems like voting and auction, exams have not been designed with security in mind. This is problematic since the recent growth in use of exam protocols has not been followed, nor preceded, by a rigorous understanding and analysis of their security. Although there are recent proposals for exam protocols with a security-by-design approach [HP10, CRHJDJ06, HPC04, BCCKR11], no formal analyses have been conducted against these proposals. Since almost all the existing exam protocols normally assume trusted authorities, only a small set of requirements, namely the ones concerning authentication of candidates, are usually considered in the analysis. As already noted in the previous chapter, exam authorities can be corrupted as well as can candidates, and exam protocols should consider a larger set of requirements including privacy.

We identify and formalise a number of authentication and privacy requirements for exams. Although we find them highly desirable out of personal experience and discussions with colleagues, the list of requirements is not meant to be universal or exhaustive. It means that certain exam protocols might demand additional requirements. However, we consider our set of requirements to be *fundamental* as similar requirements can be found in other independent works [Wei05, FOK+98], still only informally.

From the formalisation of the requirements, we build a framework for the authentication and privacy analysis of exam protocols. The framework can be used to analyse different types of exams as corroborated by the three protocols analysed in this dissertation. The requirements are specified in the applied π-calculus [AF01], a process calculus that extends the π-calculus with support for a wide variety of cryptographic properties.

Outline of the chapter. Section 3.1 examines different formal approaches for the analysis of security protocols. Section 3.2 outlines the basic constituents of the applied π–calculus. Section 3.3 introduces the formal framework by specifying the formal model of an exam. Section 3.4 continues the description of the framework by specifying authentication and privacy requirements for exam in the applied π–calculus. Section 3.5 contains the description and analysis of the Huszti-Pethő [HP10] exam protocol, which validates the framework. Moreover, the section discusses findings and proposes fixes to the protocol. Finally, Section 3.6 concludes the chapter.

3.1 Related Work

Security protocols have been historically analysed with two different approaches, one based on symbolic model, and one based on computational complexity theory. Symbolic analysis methods for protocol analysis find their root in the seminal works of Needham and Schroeder [NS78] and Dolev and Yao [DY83], which assume perfect cryptography and an unbounded active attacker who controls the entire network. Formal logic and automated tools based on the symbolic model have been used successfully to analyse security protocols.

Methods based on computational complexity theory have been initially developed by Goldwasser and Micali [GM84]. Analysis in the computational model usually see the attacker as a polynomial probabilistic Turing machine. Such analysis is deemed to be more realistic because avoids the perfect cryptography assumption, thus provides more insights about vulnerabilities of security protocols: an attack in the symbolic model leads to an attack in the computational model, while the contrary is not true in general. However, methods based on computational complexity theory are harder to mechanise (only recently mechanised tools have been proposed to assist manual proofs [Bla08, BGZB09]) and are not suitable for automation. Proofs are mostly manual and difficult as they involve to reason about probability and computational complexity, hence prone to human errors. We thus choose to develop the formal framework for exam using the symbolic model.

The symbolic model. Several symbolic techniques have been proposed over the last 25 years. Merritt [Mer83] proposed to describe protocols by rewrite rules. Burrows *et al.* [BAN90] introduced the so-called *BAN logic* to reason about authentication goals. Meadows [Mea96] proposed a language based on events for specifying security protocols, and introduced the *NRL Analyzer*, a model checker capable to verify authentication and secrecy goals. Ryan and Schneider [RS00] pioneered the idea of using process algebras for the analysis of security protocols. In Hoare's CSP [Hoa78, Sch98] the protocol's principals

are naturally modelled as processes, while security goals are modelled as reachability properties. Paulson [Pau98] introduced the *Inductive Approach* wherein principals are modelled as a set of rules that inductively define an unbounded set of traces. A protocol guarantees a security goal if the goal inductively holds for all possible traces. Thayer [FÁ99] proposed the strand spaces approach, which features an intuitive way to reason about traces generated by the security protocols: a strand is a sequence of events in which a protocol's principal may participate. Abadi and Gordon [AG97] introduced the spi-calculus, a process algebra that extends the π-calculus [MPW92] with explicit representation of cryptographic operations. The next chapter extensively discusses the applied π–calculus [AF01], which extends further the spi-calculus with a richer algebra for the modelling of cryptographic primitives.

Security requirements. The key requirements at the heart of information security are authentication and privacy. The notion of authentication has found different flavours in the literature. Gollmann [Gol96] argued that capturing the notion of authentication is difficult. Lowe [Low97] taxomised authentication in *Aliveness*, *Weak agreement*, *Non-injective agreement*, and *Injective agreement*. Despite the different interpretations, most of the approaches agree that authentication is a correspondence property: if the principal A accepts a message from B, then the principal B has actually sent that message to A. Typically, authentication can be captured by introducing *events* into the specification of protocol roles. An event explicitly signals that a principal has completed part of a run of the protocol, and what data has used in that run. The placement of events into the protocol, the corresponding data, and the relationship between events can thus capture a precise authentication goal.

Several definitions for privacy have been proposed in the literature, such as secrecy, anonymity, unlinkability, and untraceability. Dolev and Yao [DY83] specified secrecy as a reachability property, meaning that a secret is not made available or disclosed to the attacker. Schneider and Sidiropoulos [SS96] formalised anonymity in CSP as the impossibility for an attacker to link a principal with a message. Deursen *et al.* [vDMR08] clarified that unlinkability considers whether links can be established between sender and receiver principals, while untraceability considers whether different communications can be attributed to the same principal. Ryan and Schneider [RS01] observed that the notion of non-interference [GM82] and bisimulation can be used to express security requirements. On this vein, several formal definitions of privacy have been proposed as equivalence-based properties. For example, *observational equivalence* states that an observer cannot distinguish any difference between two processes, although they might perform different computations on different data.

Tools. There are several tools that support the automatic analysis of authentication and privacy. FDR [Ros97] is the most popular model checker for CSP, and contributed to the discovery of Lowe's attack. AVISPA [ABB+05] combines four techniques to analyse reachability properties (authentication and secrecy), and recently a security flaw in Google implementation of SAML 2.0 Single Sign-On Authentication was discovered [ACC+08] using that tool. ProVerif [Bla01] is an automatic protocol analyser that can prove reachability and equivalence-based properties. The input language of ProVerif is the applied π–calculus,

which the tool automatically translates to Horn clauses. ProVerif proved to be one of the automatic analysers with the best performances [CLN09]. Moreover, it allows user-defined equational theories that extend security models with algebraic properties in order to weaken the perfect cryptography assumption. We thus choose formalise our security requirements in the applied π–calculus and analyse the exam protocols with ProVerif.

Comparison with voting and auctions. Few works [Wei05, FOK+98] list a number of security requirements for exams, still only informally. Similar domains such as voting and auction have seen significant advances during the last few years, mostly in formal definitions of privacy requirements. Novel voting protocols [RS06, Adi08] have been formally analysed for a family of privacy requirements, such as ballot privacy, receipt-freeness, and coercion-resistance [DLL12, BHM08, DKR09], while auction protocols have been analysed for privacy and fairness requirements [DJP10, DLL13, DJL13]. It can be observed that only a few of security requirement definitions proposed in voting and auction domains are similar to ones we introduce for exams. For example, answers originated by eligible candidates should be marked in an exam. In the same way, only ballots cast by eligible voters should be recorded in a voting system, and only offers submitted by eligible bidders should be considered in an auction.

The requirement of mark privacy for exams is intuitively close to the definitions of ballot privacy for voting and losing bid privacy for auctions. However, it can be noticed a subtle difference: voting usually requires ballot privacy also towards the voting authority, while a mark eventually needs to be associated to the candidate usually by means of an exam authority.

Other requirements have fundamental differences. In exams, the association between an answer and its author should be preserved — even in the presence of colluding candidates. Conversely, vote authorship is not a requirement for voting, in fact unlinkability between voter and vote is a desired property. A peculiar requirement for exam is to keep the questions secret until the testing phase. In voting, the list of candidates is public, while in auctions the list of goods is normally known to bidders. Moreover, exams may require anonymous marking, namely answers are marked while ignoring their authors. This signifies a sort of fixed-term anonymity since each mark eventually needs to be assigned to the corresponding candidate.

3.2 The Applied π-calculus

The applied π-calculus [AF01] is a formal language for the description and analysis of security protocols, in which principals are represented as processes. Its syntax consists of *names*, *variables*, and *signatures*. The latter are function symbols each with an arity. Names represent channels and data, while function symbols represent cryptographic primitives such as encryption, decryption, digital signature, and hash functions. Function symbols applied to names and variables generate *terms*. Tuples of arity l, such as $n_1, \ldots, n_l$, can be abbreviated in $\tilde{n}$.

Equational theories. Whereas the π-calculus supports only a fixed set of cryptographic primitives, the applied π-calculus allows one to model user-defined

primitives by means of *equational theories*. An equational theory E describes the equations that hold on terms built from the signature. Terms are related by an equivalence relation $=$ induced by E. For instance, the equation $dec(enc(m, pk(k)), k) = m$ models an asymmetric encryption scheme. The term m is the message, the term k is the secret key, the function $pk(k)$ models the corresponding public key, the term enc models the encryption function, and the term dec models the decryption function.

Processes. The grammar for *plain* processes is outlined in Figure 3.1. The null process 0 does nothing; the process $P|Q$ is the parallel composition of processes P and Q; the process $!P$ behaves as an unbounded number of copies of processes P running in parallel; the process $\nu n.P$ generates a new *private* name n, then behaves like P; the conditional process 'if $m = m'$ then P else Q' behaves like the process P if $m = m'$ and like the process Q otherwise. For brevity, one can omit sub-term 'else Q' when Q is 0; the process $in(u, x).P$ awaits for an input from channel u, then behaves as the process P with the received message replacing the variable x; Finally, the process $out(u, m).P$ outputs the message m on the channel u, then behaves as the process P. For brevity, one can omit $.P$ when the process P is 0.

$P, Q, R ::=$	plain processes
0	null process
$P\|Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction (new)
if $m = m'$ then P else Q	conditional
$in(u, x).P$	message input
$out(u, m).P$	message output

Figure 3.1: The grammar for plain processes in the applied π-calculus

The grammar for *extended* processes is outlined in Figure 3.2. Extended processes model the knowledge exposed to the attacker. An *active substitution* $\{^m/_x\}$ is a process that replaces the variable x with the term m. We refer to a substitution also with σ. We use $m\sigma$ to refer to the result of applying σ to m. The sets $fv(A)$, $bv(A)$, $fn(A)$ and $bn(A)$ respectively include free variables, bound variables, free names, and bound names of the process A. An extended process is *closed* if all variables are bound or defined by an active substitution.

The definition of *corrupted process* [DKR06] is useful to model corrupted principals who actively collaborate with the attacker.

$A, B, C ::=$	extended processes
P	plain process
$A\|B$	parallel composition
$!P$	replication
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{^m/_x\}$	active substitution

Figure 3.2: The grammar for extended processes in the applied π-calculus

The definition outlined below specifies how to transform a process into a corrupted process. This transformation is based on two channels c_1 and c_2, which the process uses to receive and send data to the attacker.

Definition 7 (Corrupted process P^{c_1,c_2}) *Let P be a plain process and c_1, c_2 be two channel names such that $c_1, c_2 \notin fn(P) \cup bn(P)$. The* corrupted process *P^{c_1,c_2} is defined as follows:*

- $0^{c_1,c_2} \mathrel{\hat{=}} 0$,
- $(P|Q)^{c_1,c_2} \mathrel{\hat{=}} P^{c_1,c_2}|Q^{c_1,c_2}$,
- $(!P)^{c_1,c_2} \mathrel{\hat{=}} !P^{c_1,c_2}$,
- $(\nu n.P)^{c_1,c_2} \mathrel{\hat{=}} \nu n.out(c_1, n).P^{c_1,c_2}$ *if n is a name of base type, otherwise* $(\nu n.P)^{c_1,c_2} \mathrel{\hat{=}} \nu n.P^{c_1,c_2}$,
- $(\mathit{if}\ m = m'\ \mathit{then}\ P\ \mathit{else}\ Q)^{c_1,c_2} \mathrel{\hat{=}} in(c_2, x).\mathit{if}\ x = \mathit{true}\ \mathit{then}\ P^{c_1,c_2}\ \mathit{else}\ Q^{c_1,c_2}$ *where x is a fresh variable and true is a constant,*
- $(in(u, x).P)^{c_1,c_2} \mathrel{\hat{=}} in(u, x).out(c_1, x).P^{c_1,c_2}$ *if x is a variable of base type, otherwise* $(in(u, x).P)^{c_1,c_2} \mathrel{\hat{=}} in(u, x).P^{c_1,c_2}$,
- $(out(u, m).P)^{c_1,c_2} \mathrel{\hat{=}} in(c_2, x).out(u, x).P^{c_1,c_2}$, *where x is a fresh variable.*

A *frame* is an extended process built from 0 and active substitutions of the form $\{^m/_x\}$ by parallel composition and restriction. We use Φ and Ψ to range over frames. The domain $dom(\Phi)$ of a frame Φ is the set of the variables for which Φ defines a substitution. Every extended process A can be mapped to a frame $\Phi(A)$ by replacing every plain process in A with 0. The frame $\Phi(A)$ can be seen as a representation of the knowledge of the process to its environment.

Finally, a *context* is an extended process C with a hole, written $C[\_]$. It can be used to represent the environment in which the process is run.

Reachability and Correspondence properties

In the applied π–calculus, secrecy can be modelled as a reachability property. The secrecy of a term m is preserved if an attacker, defined as an arbitrary process, cannot construct m from any run of the protocol. The definitions of *name distinct*, and *reachability-based secrecy* [RS11] models secrecy. A name-distinct process signifies that the names mentioned in a term appear unambiguously in the process either as free or bound names. The definition of reachability-based secrecy says that an attacker cannot build a process A that can output the secret term m.

Definition 8 (name-distinct for $\tilde{m}$) *A plain process P is* name-distinct for *a set of names $\tilde{m}$ if $\tilde{m} \cap fn(P) \cap bn(P) = \emptyset$ and for each name $n \in \tilde{m} \cap bn(P)$ there is exactly one restriction νn in P.*

Definition 9 (Reachability-based secrecy) *A plain process P that is name-distinct for the names mentioned in the term m preserve* reachability-based secrecy *if there is no plain process A such that $(fn(A) \cup bn(A)) \cap bn(P) = \emptyset$ and $P|A$ can output m.*

In the applied π–calculus, authentication can be defined using *correspondence assertions* [WL93]. An event e is a message emitted into a special channel that is not under the control of the attacker. Events may contain arguments $M_1, ...M_n$, which are never revealed to the attacker. Events do not change the behaviour of the process in which they are located, but normally flag important steps in the execution of the protocol. To model correspondence assertions, we annotate processes with events such as $\mathsf{e}\langle M_1, ...M_n\rangle$ and reason about the relationships ($\rightsquigarrow$) between events and their arguments in the form *"if an event* $\mathsf{e}\langle M_1, ...M_n\rangle$ *has been executed, then event* $\mathsf{e}'\langle N_1, ...N_n\rangle$ *has been previously executed"*, which is formalised as the following definition.

Definition 10 (Correspondence assertion) *A* correspondence assertion *is a formula of the form* $\mathsf{e}\langle M_1, ...M_i\rangle \rightsquigarrow \mathsf{e}'\langle N_1, ...N_j\rangle$.

By adding the keyword *inj*, it is possible to model an injective correspondence assertion, which signifies that *"if an event* $\mathsf{e}\langle M_1, ...M_n\rangle$ *has been executed, then a distinct earlier occurrence of event* $\mathsf{e}'\langle N_1, ...N_n\rangle$ *has been previously executed"*.

Definition 11 (Injective correspondence assertion) *An* injective correspondence assertion *is a formula of the form* $\mathsf{e}\langle M_1, ...M_i\rangle \rightsquigarrow inj\ \mathsf{e}'\langle N_1, ...N_j\rangle$.

Authentication is only one of the requirement that can be modelled by correspondence assertions. Correspondence assertions can, for instance, capture also verifiability requirements, as we shall see in chapter 4.

Observational Equivalence

The notion of observation equivalence can capture privacy requirements. Informally, two processes are observational equivalent if an observer cannot distinguish the processes despite they might handle different data or perform different computations. To formalise observational equivalence, we first introduce the notion of *internal reduction* ($\rightarrow$), which captures the evolution of a process with respect to communication and conditionals as:

- $out(c, x).P | in(c, x).Q \rightarrow P|Q$;
- if $n = n$ then P else $Q \rightarrow P$;
- if $l = m$ then P else $Q \rightarrow Q$, where L and m are not equivalent.

Definition 12 (Observational Equivalence) Observational equivalence *($\approx$) is the largest symmetric relation $\mathcal{R}$ on extended processes such that $A\ \mathcal{R}\ B$ implies:*

1. *if $A \rightarrow^* C[out(c, M).P]$, then $B \rightarrow^* C[out(c, M).P]$;*
2. *if $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B';*
3. *$C[A]\ \mathcal{R}\ C[B]$ for all context $C[\_]$.*

The relation $\rightarrow^*$ expresses the transitive and reflexive closure of the relation $\rightarrow$. Definition 12 says that two processes A and B are observational equivalent if: 1. the process A evolves to a process that can output on channel c, also B

can evolve to a similar process; 2. if A evolves to some process A', also B can evolve to some process B', and A' and B' are observational equivalent; 3. for all contexts, $C[A]$ and $C[B]$ are observational equivalent.

The definition of observational equivalence is impracticable because requires the quantification over contexts. To avoid quantification over contexts, we first introduce the definitions of *equality of terms* and *static equivalence*. The latter captures the static part of observational equivalence as it only examines the current state of the processes. Then, we formalise *labelled bisimilarity*, which captures the dynamic behaviour of the processes and is equivalent to observational equivalence. In fact, Abadi and Fournet [AF01] proved that observational equivalence and labelled bisimilarity coincide.

Definition 13 (Equality of Terms) *Two terms m and m' are equal in the frame Φ, written $(m = m')\Phi$, if $\Phi \equiv \nu\tilde{n}.\sigma$, $m\sigma = m'\sigma$ and $\{\tilde{n}\} \cap (fn(m) \cup fn(m')) = \emptyset$, for some names $\tilde{n}$ and some substitution σ.*

Definition 14 (Static Equivalence) *Two closed frames Φ and Ψ are* statically equivalent*, written $\Phi \approx_s \Psi$, if $dom(\Phi) = dom(\Psi)$, and for all terms m and m' we have that $(m = m')\Phi$ if and only if $(m = m')\Psi$. Two extended processes A and B are statically equivalent, written $A \approx_s B$ if their frames are statically equivalent.*

In fact, two processes are statically equivalent if all their previous operations gave the same results so that they cannot be distinguished from the messages they exchange with the environment.

Definition 15 (Labelled Bisimilarity) Labelled bisimilarity *($\approx_l$) is the largest symmetric relation $\mathcal{R}$ on extended processes, such that $A\ \mathcal{R}\ B$ implies:*

1. *$A \approx_s B$*
2. *if $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B'*
3. *if $A \xrightarrow{\alpha} A'$, $fv(\alpha) \subseteq dom(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A'\mathcal{R}B'$ for some B'.*

The relation $A \rightarrow^{\alpha} A'$ defines a labelled semantics that avoids the quantification over the contexts. It signifies that A can evolve to A' using a labelled transition. This happens when A performs an input or an output due to some interaction with the environment, and α is the label standing for the involved action.

3.3 Modelling Exams

The roles of an exam can be modelled as processes in the applied π-calculus. These processes communicate via public or private channels, and can create fresh random values, which can serve as key or nonce, for example. Processes can perform tests and cryptographic operations, which are functions on terms with respect to an equational theory describing some algebraic properties.

The threat model of an exam protocol consists of a Dolev-Yao attacker who has full control of the network, namely of the public channels. The attacker

can also inject messages of his choice into the public channels, and exploit the algebraic properties of cryptographic primitives due to an equational theory. Moreover, the ability of the attacker can be extended with corrupted principals according to Definition 7. However, the attacker has no control of private channels, which are normally used to model out-of-band communications between processes. The attacker cannot even know if any communication happens over private channels. Thus, he can eavesdrop, drop, substitute, duplicate, and delay messages that principals send one another over public channels.

Having seen the basic constituents of the applied π–calculus, we can provide the definition of *exam protocol* according the calculus.

Definition 16 (Exam protocol) *An* exam protocol *is a tuple* $(C, E, Q, K, A_1, \ldots, A_l, \tilde{n}_p)$*, where* C *is the process executed by the candidates,* E *is the process executed by the examiners,* Q *is the process executed by the question committee,* K *is the process executed by the collector,* $A_1, \ldots, A_l$ *are the processes executed by the remaining authorities, and* $\tilde{n}_p$ *is the set of private channel names.*

Note that we make explicit the examiner E, the question committee Q, and the collector K among the authority processes although this is not strictly necessary. However, it turns out to be convenient for the formalisation of our security requirements.

All the principals playing the candidate role execute the same process C. However, each principal is instantiated with different variable values, e.g., keys, identities, and answers. Similarly, each principal playing any one of the authority role (e.g., examiner, question committee, collector, etc.) executes the respective processes with different values.

Definition 17 (Exam instance) *An* exam instance *of an exam protocol given by the tuple* $(C, A_1, \ldots, A_l, \tilde{n})$ *is a closed process*
$EP = \nu\tilde{n}.(C\sigma_{id_1}\sigma_{a_1}| \ldots | C\sigma_{id_j}\sigma_{a_j}| E\sigma_{id'_1}\sigma_{m_1}| \ldots | E\sigma_{id'_k}\sigma_{m_k}| Q\sigma_q| K\sigma_{test}| A_1| \ldots |A_l)$*, where*

- $\tilde{n}$ *is the set of all restricted names, including the private channels;*
- $C\sigma_{id_i}\sigma_{a_i}$*'s are the processes run by the candidates, where the substitutions* σ_{id_i} *and* σ_{a_i} *specify the identity and the answers associated with the* i^{th} *candidate;*
- $E\sigma_{id'_i}\sigma_{m_i}$*'s are the processes run by the examiner authorities, where the substitution* $\sigma_{id'_i}$ *and* σ_{m_i} *specify the identity and the mark associated with the* i^{th} *examiner;*
- $Q\sigma_q$ *is the process run by the question committee authority, where the substitution* σ_q *specifies the exam questions;*
- $K\sigma_{test}$ *is the process run by the collector authority, where the substitution* σ_{test} *associates a test with an examiner for marking;*
- A_i*'s are the processes run by the remaining exam authorities.*

Definitions 16 and 17 capture the levels of detail and abstraction advocated in chapter 2. The instance of an exam protocol can be customised by making processes A_i explicit. For example, the exam instance can be expanded with processes that model a mixnet for the generation of test pseudonyms (see chapter 5), or a bulletin board that publishes the results of an exam (see chapter 6).

As we shall see later, Definition 17 allows us to specify a considerable number of security requirements and is suitable for different types of exam protocols. Moreover, it equally supports either machine or human examiners as principals that mark answers.

3.4 Security Requirements

We identify and formalise a set of fundamental *authentication* and *privacy* requirements in the applied π-calculus. This set is not meant to be comprehensive, but it includes the basic security requirements that an exam protocol is normally expected to guarantee as corroborated in the literature [Wei05, BGL13c, FOK+98]. However, our set can be extended with additional security requirements.

We introduce five authentication and five privacy requirements. The authentication requirements capture the associations between the candidate's identity, the answer, and the mark being preserved through all the exam phases. When authentication holds there is no loss, no injection, and in general no manipulation of the exam tests from preparation to notification. The five privacy requirements aim to capture secrecy of marks, and anonymity of tests and examiners.

3.4.1 Authentication

To model authentication requirements as correspondence properties, it is necessary to define a number of relevant events. Events normally need to agree with some arguments to capture authentication. Thus, we introduce the terms that serve as arguments in our events as follows.

- *id_c* refers to the identity of the candidate;
- *ques* denotes the question(s) of the test;
- *ans* denotes the answer of a test;
- *mark* denotes the mark assigned to the test;
- *id_e* refers to the identity of the examiner;
- *id_test* refers to the identifier of the test.

The terms outlined above intuitively relates to the substitutions introduced in Definition 17. Their definitions are abstract enough to capture different exams. For example, the term *id_test* may coincide with the identity of the candidate if the exam requires no blind marking, or may be a pseudonym to if the exam requires anonymous marking.

We define a list of six events that allow to specify five fundamental authentication requirements for exams. We stress that the list can be further extended to accommodate any additional requirements.

- `registered`$\langle id\_c \rangle$ means that the authority considers the candidate id_c registered for the exam. The event is inserted into the process of authority at the location where the registration of the candidate id_c concludes.

- `submitted`$\langle id\_c, ques, ans, id\_test \rangle$ means that the candidate id_c considers the test id_test, which consists of question $ques$ and answer ans, submitted for the exam. The event is inserted into the process of the candidate at the location where the test is sent to the collector.

- `collected`$\langle id\_c, ques, ans, id\_test \rangle$ means that the collector accepts the test id_test, which originates from the candidate id_c. The event is inserted into the process of the collector at the location where the test is considered as accepted.

- `distributed`$\langle id\_c, ques, ans, id\_test, id\_e \rangle$ means that the collector considers the test id_test, which originates from the candidate id_c, associated with the examiner id_e for marking. The event is inserted into the process of the collector at the location where the test is distributed to the examiner.

- `marked`$\langle ques, ans, mark, id\_test, id\_e \rangle$ means that the examiner id_e considers the test id_test, which consists of question $ques$ and answer ans, evaluated with $mark$. The event is inserted into the process of the examiner at the location where the test is marked.

- `notified`$\langle id\_c, mark \rangle$: means that the candidate id_c accepts the mark $mark$. The event is inserted into the process of the candidate at the location where the mark is considered as accepted.

These events mark important steps of an exam protocol, and some can be associated with the phases of an exam. The event `registered` normally concludes the preparation phase, while `collected` concludes the testing phase. The event `distributed` begins the marking phase, which the event `marking` concludes. Finally, the event `notified` concludes the notification phase and the exam. Note that these events implicitly refer to the same exam session. However, one might want to parameterise all the events with a common term in order to distinguish among exam sessions.

The first authentication requirement we consider is *Candidate Authorisation*, which concerns preparation and testing. Informally, we want to capture the requirement that only registered candidates can take the exam. More specifically, the requirement says that if a candidate submits her test, then the candidate was correctly registered for the exam. This can be formalised as:

Definition 18 (Candidate Authorisation) *An exam protocol ensures* Candidate Authorisation *if for every exam process EP*

$$\texttt{submitted}\langle id\_c, ques, ans, id\_test \rangle \rightsquigarrow inj\texttt{registered}\langle id\_c \rangle$$

on every execution trace.

This requirement is modelled as injective correspondence assertion because the exam should consider only one submission per registered candidate.

The second authentication requirement that we advance is *Answer Authenticity*, which concerns testing. This requirement states that the collector should consider only answers that candidates actually submitted, and that the contents of each collected test are not modified after submission. It says that a test must be bound to a candidate identity. A way to enforce this would be to only give a test to a candidate after she inserts in the test the same details that authenticated her. This candidate becomes the test assignee. With exams that are not computer assisted, for example, an authority can check that the candidate writes down the right details on the test, or the authority can write them down personally. The requirement implies that two candidates will be unable to get tested on each other's questions, something that could be desirable if they found their respective questions too difficult. Moreover, it should be considered only one test from each candidate, namely every time the collector process emits `collected`, there is a distinct earlier occurrence of the event `submitted` that satisfies the relationship between their arguments. This is enforced by the injective formula:

Definition 19 (Answer Authenticity) *An exam protocol ensures* Answer Authenticity *if for every exam process EP*

$$\texttt{collected}\langle id\_c, ques, ans, id\_test\rangle \rightsquigarrow inj\texttt{submitted}\langle id\_c, ques, ans, id\_test\rangle$$

on every execution trace.

The third requirement is *Test Origin Authentication* and concerns preparation and testing. Informally, it says that the collector should accept only tests that originate from registered candidates. This requirement should be modelled as an injective agreement to enforce that only one test from each registered candidate is actually collected. This can be formalised as:

Definition 20 (Test Origin Authentication) *An exam protocol ensures* Test Origin Authentication *if for every exam process EP*

$$\texttt{collected}\langle id\_c, ques, ans, id\_test\rangle \rightsquigarrow inj\texttt{registered}\langle id\_c\rangle$$

on every execution trace.

The forth authentication requirement is *Test Authenticity* and concerns testing and marking. Since the collector distributes the tests possibly among different examiners, Test Authenticity insists that the examiner only marks the tests intended for him. Moreover, the contents of each test should not be modified until after the tests are marked by the examiner. This requirement can be modelled as injective agreement:

Definition 21 (Test Authenticity) *An exam protocol ensures* Test Authenticity *if for every exam process EP*

$$\begin{gathered}\texttt{marked}\langle ques, ans, mark, id\_test, id\_e\rangle \rightsquigarrow \\ inj\texttt{collected}\langle id\_c, ques, ans, id\_test\rangle \cup \\ inj\texttt{distributed}\langle id\_c, ques, ans, id\_test, id\_e\rangle\end{gathered}$$

on every execution trace.

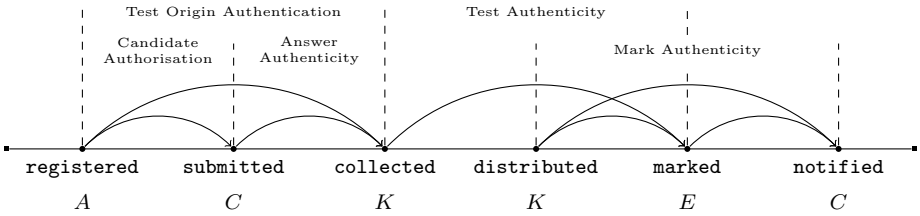

Figure 3.3: A general view of authentication requirements for exams

The last requirement is *Mark Authenticity*, which concerns marking and notification. It prescribes that the candidate should receive the mark assigned to her test by the examiner. Moreover, the examiner who evaluated the test should be the one chosen by the collector. In other words, if the candidate accepts the mark, then the examiner, which was appointed by the collector to evaluate the candidate's test, assigned that mark. This can be formalised as:

Definition 22 (Mark Authenticity) *An exam protocol ensures* Mark Authenticity *if for every exam process EP*

$$\begin{gathered}\texttt{notified}\langle id\_c, mark\rangle \rightsquigarrow \\ inj\texttt{distributed}\langle id\_c, ques, ans, id\_test, id\_e\rangle \cup \\ inj\texttt{marked}\langle ques, ans, mark, id\_test, id\_e\rangle\end{gathered}$$

on every execution trace.

Remark. It can be observed that the combination of these requirements produce novel requirements. If an exam protocol guarantees Candidate Authorisation and Answer Authenticity, then the protocol also guarantees Test Origin Authentication, namely the tests submitted by registered candidates are actually collected. Conversely, a protocol that guarantees Test Origin Authentication may guarantee neither Candidate Authorisation nor Answer Authenticity.

If we consider a requirement in a certain phase of the exam, we cannot infer anything about other phases. For example, Mark Authenticity signifies that the candidate is notified with the mark delivered by the examiner on the test provided by the collector. However, the test provided by the collector may contain a different answer with respect to the answer the candidate submitted at testing. Only if the exam protocol also guarantees Answer Authenticity and Test Authenticity, then the contents of the tests are identical. Moreover, Mark Authenticity does not signify that a mark is computed correctly.

In summary, an exam protocol that ensures all the requirements outlined above preserves the association between candidate identity, mark, and test, including question and answer, through all the phases of the exam. The relationships between authentication requirements with respect to exam run and principals are outlined in Figure 3.3. By looking at this figure, it can be seen that the stated requirements produce an ordered sequence of events. It can be noted that there is no requirement that relates directly the events `collected` and `distributed`. We have not specified the requirement *"the collector distributes*

the accepted tests". Such requirement is usually enforced by the sequential execution of the collector process, since both events belong to the same process. Moreover, it always holds if the common arguments of the two events are derived from the same source, for example, if the common arguments are build from the same message input. In general, this and other requirements get more interesting if events reflect tasks that are executed by different roles, according to the levels of detail and abstraction of the exam protocol. Hence, an exam protocol that specifies a huge number of tasks, might allow to express more events thus novel authentication requirements.

3.4.2 Privacy

To model privacy requirements as equivalence properties, we use the definition of labelled bisimilarity ($\approx_l$), which was defined in Section 3.2.

We introduce two notations to make clear the requirements. First, we denote with "$EP_I[\_]$" the context of the process EP pruned of identities that appear in the set I. For example, the process $\nu\tilde{n}.(\_|\_|\ C\sigma_{id_3}\sigma_{a_3}|\ldots|C\sigma_{id_j}\sigma_{a_j}|$ $E\sigma_{id'_1}\sigma_{m_1}|\ldots\ |E\sigma_{id'_k}\sigma_{m_k}|Q\sigma_q|K\sigma_{dist}|A_1|\ldots|A_l)$ can be concisely written as $EP_{\{id_1,id_2\}}[\_]$. Such compact notation is useful to specify and focus exactly on the processes concerned by the requirement. For example, we can write $EP_{\{id_1,id_2\}}[C\sigma_{id_1}\sigma_{a_1}|C\sigma_{id_2}\sigma_{a_2}]$ to reason about candidates id_1 and id_2 without repeating the entire exam instance.

Second, we denote with "$EP|_{\mathtt{e}}$" the process EP pruned of the code that follows the event e. For example, the process $EP|_{\texttt{marked}}$ considers an exam instance that terminates at marking, namely after the event `marked` is emitted. This notation is useful to capture fixed-term requirements such as anonymous marking, which is intended to hold until after the marking, but is eventually falsified at notification when the mark is assigned to the candidate.

The first privacy requirement we consider is *Question Indistinguishability*. This requirement says that the questions are not revealed until the testing phase begins. Thus, it is a fixed-term requirement that sees the exam process ending with the preparation phase.

Definition 23 (Question Indistinguishability) *An exam protocol ensures* Question Indistinguishability *if for any exam process EP and any questions q_1 and q_2*

$$EP[Q\sigma_{q_1}]|_{\texttt{registered}} \approx_l EP[Q\sigma_{q_2}]|_{\texttt{registered}}$$

Question Indistinguishability states that two processes with different questions have to be observationally equivalent until after the preparation phase. Note that this requirement is more stringent than reachability-based secrecy because the attacker should not be able to distinguish whether the exam will use q_1 or q_2 despite he knows both the questions in advance. For instance, the attacker cannot say whether the questions of the current exam are similar to the questions of the previous exam, which are on attacker's knowledge. The analysis of Question Indistinguishability requires the question committee to be honest otherwise they could reveal the questions to the attacker making the requirement useless. However, it is particularly interesting to consider a scenario with other corrupted roles. For example, candidates might be interested to know the questions in advance. Such scenario can be explicitly captured by replacing

honest candidates with corrupted ones using the definition of corrupted process, which is defined in Section 3.2. Assuming candidate id_1 being corrupted, we obtain

$$EP_{\{id_1\}}[(C\sigma_{id_1}\sigma_{a_1})^{c_1,c_2}|Q\sigma_{q_1}]|_{\texttt{registered}} \approx_l EP_{\{id_1\}}[(C\sigma_{id_1}\sigma_{a_1})^{c_1,c_2}|Q\sigma_{q_2}]|_{\texttt{registered}}$$

The next requirement is *Anonymous Marking*, which covers preparation, testing, and marking. This requirement signifies that the examiner marks a test while ignoring its author, namely an anonymous test. It is a clear contribution to the fairness of the marking. As it stands, the requirement insists on anonymity only until the point that the examiner affixes a mark. Anonymous Marking can be specified as two exam instances in which the processes of two candidates who swap their answers cannot be distinguished until after the end of the marking phase.

Definition 24 (Anonymous Marking) *An exam protocol ensures* Anonymous Marking *if any exam process EP, any two candidates id_1 and id_2, and any two answers a_1 and a_2*

$$EP_{\{id_1,id_2\}}[C\sigma_{id_1}\sigma_{a_1}|C\sigma_{id_2}\sigma_{a_2}]|_{\texttt{marked}} \approx_l EP_{\{id_1,id_2\}}[C\sigma_{id_1}\sigma_{a_2}|C\sigma_{id_2}\sigma_{a_1}]|_{\texttt{marked}}$$

In other words, Anonymous Marking says that the process where candidate id_1 submits a_1 and candidate id_2 submits a_2 is indistinguishable to the process where candidate id_1 submits a_2 and candidate id_2 submits a_1. It prevents the attacker to obtain the identity of the candidate who submits a certain answer before the marking ends. Candidate

Similarly to Question Indistinguishability, it is interesting to consider corrupted principals in the analysis of Anonymous Marking, which means that nobody knows who submitted a test while this is being marked, except the official author of the test. An implication is that test anonymity during marking will even resist collusion of the examiner with other authorities and candidates. Again, the definition of corrupted process can model corrupted examiners and authorities. The definition can be also used to specify corrupted candidates, however the candidates id_1 and id_2 who submit two different answers have to be honest. This avoids the corner case in which all candidates but one reveal their answers to the attacker, who can easily associate the remaining answer with the honest candidate and thus trivially violate the requirement.

We now consider the requirement of *Anonymous Examiner*, which concerns all the phases of an exam. In fact, an exam could require examiner anonymity forever to prevent bribing or coercion. Thus, the requirement of Anonymous Examiner says that no candidate knows which examiner marked their tests. This can be formalised as:

Definition 25 (Anonymous Examiner) *An exam protocol ensures* Anonymous Examiner *if for any exam process EP, any two candidates id_1 and id_2, any two examiners id'_1 and id'_2, any two marks m_1 and m_2, and two associations $test_1$ and $test_2$*

$$EP_{\{id_1,id_2,id'_1,id'_2\}}\ [C\sigma_{id_1}\sigma_{a_1}|C\sigma_{id_2}\sigma_{a_2}|E\sigma_{id'_1}\sigma_{m_1}|E\sigma_{id'_2}\sigma_{m_2}|K\sigma_{test_1}] \approx_l EP_{\{id_1,id_2,id'_1,id'_2\}}\ [C\ \sigma_{id_1}\ \sigma_{a_1}|C\sigma_{id_2}\sigma_{a_2}|E\sigma_{id'_1}\sigma_{m_2}|E\sigma_{id'_2}\sigma_{m_1}|K\sigma_{test_2}]$$

where

- σ_{test_1} *associates the test of candidate* id_1 *to examiner* id'_1 *and the test of candidate* id_2 *to examiner* id'_2*;*
- σ_{test_2} *associates the test of candidate* id_1 *to examiner* id'_2 *and the test of candidate* id_2 *to examiner* id'_1*.*

Thus, Anonymous Examiner states that a process in which the examiner id'_1 evaluates the test of candidate id_1 while the examiner id'_2 evaluates the test of candidate id_2 is indistinguishable to the process in which the examiner id'_1 evaluates the test of candidate id_2 while the examiner id'_2 evaluates the test of candidate id_1. Note that the two marks σ_{m_1} and σ_{m_2} are swapped on examiner processes to ensure that each test is evaluated with the same mark in both cases. In the field of peer review systems, this requirement is known as *blind review*. The requirement of *double-bind review* instead refers to a peer review system that ensure both Anonymous Examiner and Anonymous Marking, namely anonymity is provided to both authors and examiners. However, peer review systems usually assume that the collector knows which examiner evaluates a test, while other systems may not. To ensure a stronger version of Anonymous Marking it is possible to model corrupted collectors, candidates, and any other principal, provided that examiners id'_1 and id'_2 are honest. This would avoid the corner case in which an examiner reveals the mark to the attacker, a case that would trivially violate the requirement.

The requirement of *Mark Privacy* concerns all phases of an exam. It states that the mark ultimately attributed to a candidate is treated as valuable personal information of the candidates. More specifically, no one learns the marks, besides the examiner, the concerned candidate, and the authority responsible for the notification. This means that the marks cannot be public.

Definition 26 (Mark Privacy) *An exam protocol ensures* Mark Privacy *if for any exam process* EP *and any two marks* m_1 *and* m_2

$$EP_{\{id'\}}[E\sigma_{id'}\sigma_{m_1}] \approx_l EP_{\{id'\}}[E\ \sigma_{id'}\sigma_{m_2}].$$

The definition of Mark Privacy means that a process in which the examiner id' assigns the mark m_1 to an answer cannot be distinguished from a process in which the same examiner assigns a different mark m_2 to the same answer. This is a strict requirement because an exam protocol that guarantees Mark Privacy cannot publicly disclose the marks even if these cannot be associated with the corresponding candidates. In fact, the publication of the marks allows the attacker to distinguish the processes. Again, it can be assumed that some candidates and examiners are corrupted, namely collaborate with the attacker to find out the marks of other candidates. However, the examiner who assigns the different marks, the two candidates who submit the tests, and the authority in charge of the notification of the marks should be honest. Otherwise, any of these could violate the requirement by revealing the mark to the attacker.

Since Mark Privacy can be a requirement too strong to satisfy, we introduce a variant called *Mark Anonymity*. This requirement states that no one learns the association between a mark and the corresponding candidate. Intuitively, an exam protocol that publishes the list of all marks might still ensure Mark Anonymity, but not Mark Privacy. This is a common privacy requirement in scenarios like public competitions, in which marks are published and associated

with a list of pseudonyms for transparency. Mark Anonymity can be defined as follows:

Definition 27 (Mark Anonymity) *An exam protocol ensures* Mark Anonymity *if for any exam process EP, any two candidates id_1, id_2, any examiners id', any two answers a_1, a_2, two substitutions σ_{m_a} and σ_{m_b} and an association test*

$$EP_{\{id_1,id_2,id'\}}[C\sigma_{id_1}\sigma_{a_1}|C\sigma_{id_2}\sigma_{a_2}|E\sigma_{id'}\sigma_{m_a}|K\sigma_{test}] \approx_l EP_{\{id_1,id_2,id'\}}[C\sigma_{id_1}\sigma_{a_1}|C\sigma_{id_2}\sigma_{a_2}|E\sigma_{id'}\sigma_{m_b}|K\sigma_{test}]$$

where

- *σ_{test} associates the tests of both candidates id_1 and id_2 to the examiner id';*
- *σ_{m_a} attributes the mark m_1 to the answer a_1 and the mark m_2 to the answer a_2;*
- *σ_{m_b} attributes the mark m_2 to the answer a_1 and the mark m_1 to the answer a_2.*

In other words, a process in which an examiner evaluates two answers a_1 and a_2 respectively with m_1 and m_2 is indistinguishable for the attacker with a process in which the examiner evaluates the same answers but with swapped marks, namely the examiner marks a_1 and a_2 respectively with m_2 and m_1. In doing so, the authority can make the list of marks public assuming the attacker cannot associate the marks to the candidates. The analysis of Mark Anonymity requires the two concerned candidates, the examiner, and the notifier authority to be honest. Otherwise, they can simply reveal the answer and the associated mark to allow the attacker to distinguish the two case processes. Other principals can be considered corrupted. It can be noted that an exam protocol that guarantees *Mark Privacy* also guarantees *Mark Anonymity*. In fact, σ_{m_a} and σ_{m_b} defined in Mark Anonymity are special instances of σ_{m_1} and σ_{m_2} defined in Mark Privacy.

3.5 The Huszti-Pethő Protocol

We validate our formal framework with the analysis of the Huszti-Pethő [HP10] exam protocol. To the best of our knowledge, this is the first exam protocol proposed in the literature that aims to guarantee authentication and privacy requirements in presence of corrupted candidates and exam authorities.

Huszti and Pethő informally analyse their protocol with respect to six security requirements. They state the requirements informally, and each requirement contains sub requirements. For example, the requirement of *Secrecy* implicitly specifies two sub requirements as it states that *"exam questions are kept secret"* and *"only the corresponding student should know his mark"*. Table 3.1 clarifies the sub requirements and shows how to map them to the formal requirements proposed in this dissertation. By looking at the table, it can be seen that combinations of our formal requirements capture any informal requirement defined by Huszti and Pethő. For example, Huszti and Pethő's definition of Robustness, which says that questions can not be altered after submission, is captured by the

Huszti and Pethő		This dissertation
Requirement	**Description**	
Authenticity	*Only eligible students tests should be considered*	Candidate Authorisation Answer Authenticity
	It should be verified whether the exam grade is proposed by a teacher	Mark Authenticity
Anonymity	*Teachers do not know which paper belonging to which student he is correcting*	Anonymous Marking
	Students do not know who corrects their papers	Anonymous Examiner
Secrecy	*Exam questions are kept secret*	Question Indistinguishability
	Only the corresponding student should know his mark	Mark Anonymity
Robustness	*Exam questions can not be altered after submission*	Answer Authenticity Test Authenticity Mark Authenticity
Correctness	*Students are not allowed to take the same exam more than once*	Answer Authenticity
Receipt	*Students are able to make sure of the successful submission*	Answer Authenticity Test Authenticity Mark Authenticity

Table 3.1: Comparison of Huszti and Pethő's requirements with ones proposed in this dissertation.

combination of Answer Authenticity, Test Authenticity, and Mark Authenticity. We anticipate that the results of our analysis show the protocol guarantees only one of our formal requirements, but none of the six envisaged by Huszti and Pethő. The Huszti-Pethő exam protocol uses four cryptographic building blocks, namely ElGamal encryption [Elg85], zero-knowledge proof [GMR85], reusable anonymous return channel [GJ03], and a timed-release service based on Shamir's secret sharing [Sha79].

ElGamal encryption

This cryptographic primitive for public-key cryptography consists of three algorithms of *key generation*, *encryption*, and *decryption*. The key generation algorithm outputs the public key $PK = (G, q, g, h)$ and the secret key $SK = s$; G is a cyclic group of order q with generator g; s is a random value in $\mathbb{Z}_q^*$; and $h = g^s$. The encryption algorithm takes in a message m and a random value $k \in \mathbb{Z}_q^*$, and outputs the ciphertext $(g^k, m \cdot h^k)$, which is denoted with $\{m\}_{PK}$. The decryption algorithm takes as input the ciphertext $\{m\}_{PK}$ and the secret key SK, and outputs the message m. In fact, $\frac{h^k}{g^{ks}} = \frac{g^{sk}}{g^{ks}} = m$. The ElGamal encryption primitive is semantically secure assuming the decisional DiffieHellman problem is intractable.

Zero-knowledge proof

This cryptographic scheme allows a *prover* to convince a *verifier* that a given statement is true without revealing any extra information except the correctness of the statement. A zero-knowledge scheme must guarantee completeness,

soundness, and zero knowledge. Completeness means that if the statement is true, then the verifier always accepts the statement. Soundness means that if the statement is false, then the verifier always rejects the statement. Zero knowledge means that the verifier cannot get any information apart from the fact that the statement is indeed true.

Zero-knowledge schemes can be categorised in non-interactive and interactive. Interactive zero-knowledge proofs require prover and verifier to exchange at least two messages. The input of the prover is usually a challenge message sent by the verifier. In so doing, the proof is only valid for that challenge, and cannot replayed by the prover to someone else.

Non-interactive zero-knowledge proof schemes contain only the message sent by a prover to the verifier. They are simpler and more efficient than interactive schemes, hence more suitable for the inclusion in the design of cryptographic protocols. In the remainder, we only consider non-interactive schemes as zero-knowledge proof.

Reusable anonymous return channel

This cryptographic scheme implements anonymous two-way conversations between a *sender* and a *receiver* by means of a *mixer*. The mixer is implemented by a re-encryption mix network that consists of a chain of mix servers. The servers take in messages from multiple senders, randomly shuffle them, and send them to the receivers. One goal of reusable anonymous return channel is to ensure the anonymity of the sender who send a message via the mixer. Another goal is to allow the receiver to reply to the sender still guaranteeing the anonymity of the sender. Note that reusable anonymous return channel scheme aims to ensure anonymity, but not secrecy of the messages [GJ03]. As we shall see, the Huszti-Pethő protocol resorts on this primitive for both message secrecy and sender anonymity.

Reusable anonymous return channel consists of five algorithms, namely *setup*, *submission of messages*, *delivery of messages*, *submission of reply*, and *delivery of reply*. The scheme assumes a primitive for digital signature, but the authors do not specify which one. However, digital signature primitives usually employ public-key cryptography and consist of three algorithms of *key generation*, *signing*, and *verification*. Key generation outputs the secret signing key SSK and verification public key SPK. The signing algorithm takes in a message m and the signing key SSK, and outputs the signature $Sign_{SSK}(m)$. The verification algorithm takes as input the signature $Sign_{SSK}(m)$ and the verification public key SPK, and returns `true` if the signature is correct, namely the message m was actually signed with the signing key SSK.

The setup algorithm consists of mix servers jointly generating an ElGamal key pair (SK_M, PK_M), and signature keys (SSK_M, SPK_M). Sender and receiver also generate respectively the ElGamal pairs (PK_A, SK_A) and (PK_B, SK_B). The identities of sender and receiver are denoted respectively with the tags ID_A and ID_B. For example, email addresses can serve as identity tag.

The algorithm of the submission of messages is run by the sender A. It allows A to send an anonymous message m to the receiver with tag ID_B. The sender generates the triplet

$$(\{ID_A, PK_A\}_{PK_M}, \{m\}_{PK_M}, \{ID_B, PK_B\}_{PK_M})$$

1. $A \rightarrow M$: $\{ID_A, PK_A\}_{PK_M}, \{m\}_{PK_M}, \{ID_B, PK_B\}_{PK_M}$
2. $M \rightarrow B$: $Sign_{SSK_M}(\{ID_A, PK_A\}_{PK_M}), \{m\}_{PK_B}$
3. $B \rightarrow M$: $\{ID_B, PK_B\}_{PK_M}, \{m'\}_{PK_M}, Sign_{SSK_M}(\{ID_A, PK_A\}_{PK_M})$

Figure 3.4: Reusable anonymous return channel in the Alice-Bob notation

and two *proofs of knowledge* of $\{ID_A, PK_A\}$ and of $\{ID_B, PK_B\}$. Proofs of knowledge are similar to zero-knowledge schemes but guarantee only completeness and soundness. In this case, they aim to avoid the attacker to decrypt the triplets by using the mixer as an oracle. The sender outputs the triplet and the proofs.

The algorithm of delivery of messages is run by the mixer. It takes as input a batch of triplets and proofs sent from different senders. The mixer checks the proofs and then randomly shuffles the batch of triplets. Each triplet is extended with a checksum to ensure they are not separated during the shuffle. The message m is then re-encrypted with PK_B, resulting in the ciphertext $\{m\}_{PK_B}$. The mixer signs the first element of the triplet in input $\{ID_A, PK_A\}_{PK_M}$, and outputs the pair

$$(Sign_{SSK_M}(\{ID_A, PK_A\}_{PK_M}), \{m\}_{PK_B}).$$

The receiver can reply an anonymous message with a new message m' using the algorithm for submission of reply. The receiver takes in the pair $(Sign_{SSK_M}(\{ID_A, PK_A\}_{PK_M}), \{m\}_{PK_B})$, encrypts the message m' with the public key of the mixnet PK_M resulting in ${m'}_{PK_M}$, and outputs the triplet

$$(\{ID_B, PK_B\}_{PK_M}, \{m'\}_{PK_M}, Sign_{SSK_M}(\{ID_A, PK_A\}_{PK_M}))$$

and the proof of knowledge of $\{ID_B, PK_B\}$.

The algorithm of delivery of reply is similar to one of delivery of messages. The only difference is that the input consists of triplet $(\{ID_B, PK_B\}_{PK_M}$, ${m'}_{PK_M}, Sign_{SSK_M}(\{ID_A, PK_A\}_{PK_M}))$ and proof of knowledge of $\{ID_B, PK_B\}$, thus the mixer verifies only one proof of knowledge.

A succinct description of reusable anonymous return channel in the Alice-Bob notation is provided in Figure 3.4.

Timed-release service

This service is based on threshold Shamir's secret sharing, a cryptographic primitive that ensures fixed-term secrecy. A secret is shared among n servers and cannot be reconstructed unless some servers collaborate to reveal the secret. This service assumes the existence of a trusted third party, which in the Huszti-Pethő protocol is known as *registry*. The registry knows the secret, bootstraps the servers, and serves as authority to provide absolute time reference to the servers. The Huszti-Pethő protocol uses the timed-release service to deanonymise the candidate's pseudonym for notification. We do not detail this service further because in our analysis we assume the servers to be trusted.

3.5.1 Description

The Huszti-Pethő protocol specifies six roles: exam authority (EA), registry, timed-release service (NET), question committee (COM), candidate (C), examiner (E). The exam authority manages the entire exam process. Specifically, it generates the pseudonyms to anonymise candidates' tests and examiner's identities, collects the tests, distributes the tests to examiners, and notifies the marks to the candidates. The registry generates the necessary cryptographic keys for the other roles and bootstraps the timed-release service. The NET consists of the servers that implement the timed-release service, and contributes to generate and revoke the candidate pseudonyms using threshold Shamir's secret sharing. The question committee generates the questions for the exam, the candidate takes the exam, and the examiner marks the tests.

The protocol assumes that no candidate reveals their private keys to other candidates, and that invigilators supervise candidates during the testing phase. All communications take place via reusable anonymous return channels.

The protocol originally sees three stages: *registration*, *exam*, and *grading*. It can be observed that the exam stage begins with the exam authority checking the candidate's eligibility, and concludes with the examiner sending the mark to the exam authority. Thus, to match this structure with our phases, we map the registration stage to preparation, the grading stage to notification, and we divide the exam stage in testing and marking. We provide a high-level description of the protocol, which is supported by a more detailed specification in the Alice-Bob notation in Figure 3.5.

Preparation

This phase concerns the registration of both candidate and examiner, and the generation of the pseudonyms. The exam authority publishes the public parameters to identify a new exam (step 1). The question committee then signs and sends the questions, which are encrypted with the public key of the mixer implementing the reusable anonymous return channel (step 2). The mixer will publish the questions only at time of testing ($time_1$).

The registration of the examiner consists of creating a pseudonym, which is jointly generated by the exam authority and the examiner. The examiner verifies the correctness of the pseudonym by using a zero-knowledge proof (ZKP_{eq}) on the equality of the discrete logarithms with the exam authority (step 6). To enrol for an exam, the examiner sends pseudonym and subject to the exam authority (step 9), and proves the knowledge of his secret key ($ZKP_{sec}(SK_E)$). Note that the exam authority knows that the examiner is eligible for the exam, but cannot learn the examiner identity since the communication takes place via reusable anonymous return channel. Thus, at the end of examiner registration, the exam authority stores the encrypted identity of the examiner ($\{ID_E, PK_E\}_{PK_M}$) (step 10), which the exam authority will use to send the answer to the anonymous examiner at marking.

The registration of a candidate slightly differs from the registration of an examiner since the anonymity of the candidate eventually will be broken at notification, while the anonymity of examiner may last forever. The pseudonym of the candidate is jointly calculated by the exam authority, the candidate, and also the NET. The NET stores the secret values used for the generation of the

`Preparation`

1. EA publishes g and $h = g^s$
2. $COM \rightarrow EA : \{Sign_{SSK_{COM}}(question, time_1)\}_{PK_M}$
3. EA checks E eligibility, and calculates $\tilde{q} = PK_E^s$ //Examiner Registration
4. $EA \rightarrow E : \tilde{q}, g_E$
5. E calculates $q' = (\tilde{q})^\alpha$, $t = (g_E)^\alpha$, and $q = t^{SK_E}$
6. $EA \leftrightarrow E : ZKP_{eq}((q, q'), (g, h))$ //E pseudonym is (t, q, q')
7. $E \rightarrow EA : t, q, q', subject$
8. EA checks $q^s = q'$
9. $E \leftrightarrow EA : ZKP_{sec}(SK_E)$
10. EA stores ZKP data plus $\{ID_E, PK_E\}_{PK_M}$ and $subject$
11. EA checks C eligibility, and calculates $\tilde{p} = (PK_C)^s$ //Candidate Registration
12. $EA \rightarrow NET : \tilde{p}, g_C$
13. NET calculates $p' = (\tilde{p})^\Gamma$, and $r = (g_C)^\Gamma$, and stores $time$ of notification, $\tilde{p}$, and g_C.
14. $NET \rightarrow C : r, p'$
15. C calculates $p = r^{SK_C}$
16. $EA \leftrightarrow C : ZKP_{eq}((p, p'), (g, h))$ //C pseudonym is (r, p, p')

`Testing`

17. $C \rightarrow EA : r, p, p', subject$
18. EA checks $p^s = p'$
19. $C \leftrightarrow EA : ZKP_{sec}(SK_C)$
20. $EA \rightarrow C : Sign_{SSK_{COM}}(question), time_1$
21. $C \rightarrow EA : r, p, \{answer\}_{PK_M}, time_2$
22. $EA \rightarrow C : Hash(r, p, p', subject, trans_C, question, time_1, time_2, \{answer\}_{PK_M})$

`Marking`

23. $EA \rightarrow E : \{answer\}_{PK_M}$
24. $E \rightarrow EA : mark, Hash(mark, answer), [Hash(mark, answer)]^{SK_E}, verzkp$
    where $verzkp = ZKP_{eq}(Hash(mark, answer), [Hash(mark, answer)]^{SK_E}), t, q$

`Notification`

26. $EA \rightarrow NET : p'$ // Note that $r = (g_C)^\Gamma$, $p = (PK_C)^\Gamma$, $p' = (g_C)^{\Gamma s}$
27. NET calculates $p' = (\tilde{p})^\Gamma$
28. $NET \rightarrow EA : \{p', \tilde{p}\}_{PK_{EA}}$
29. EA stores $mark, Hash(mark, answer), [Hash(mark, answer)]^{SK_E}, verzkp$

Figure 3.5: The Huszti-Pethő e-exam protocol in the Alice-Bob notation

candidate pseudonym (step 13), and will use the secret values at notification to allow the exam authority to associate the candidate with the mark. Similarly to the registration of examiner, the candidate can verify the correctness of the pseudonym using a zero-knowledge proof (ZKP_{eq}) of the equality of the discrete logarithms with the exam authority being the prover (step 16).

Testing

The candidate sends the pseudonym to the exam authority (step 17) and proves the knowledge of the private key ($ZKP_{sec}(SK_C)$) (step 19). We stress that the protocol assumes the candidate does not share with other principals the private key. The exam authority checks whether the pseudonym is allowed for the exam (step 18), and then sends the questions signed by the question committee (step 20). The candidate then sends the answer (step 21) encrypted with the public key of the mixer ($\{answer\}_{PK_M}$). At marking, the mixer will re-encrypt the answer with the public key of the examiner. Thus, the exam authority cannot learn the answer submitted by the candidate. Testing concludes with the exam authority sending to the candidate a receipt (step 22), which consists of the hash of all parameters seen by the exam authority during testing, the transcription of the zero-knowledge proof ($trans_C$), and the time when the answer was submitted ($time_2$).

Marking

We recall that at preparation the exam authority stored the encrypted identities of the examiners, thus it can choose an examiner who is eligible for the exam to forwards the candidate's answer. The examiner assigns the answer with a mark (step 23), and sends it to the exam authority with a zero-knowledge proof ($verzkp$), which proves the examiner actually marked the answer (step 24).

Notification

When all the answers are marked, the NET de-anonymises the pseudonyms associated to the answers, so the exam authority can link back the pseudonym with the corresponding candidate (steps 27-28). Finally, the exam authority stores the marks and the zero-knowledge proof provided at marking by the examiner (step 29).

3.5.2 Formal Analysis of Reusable Anonymous Return Channel

Prior to verify the Huszti-Pethő protocol, we provide a formal analysis of reusable anonymous return channel. In particular, we verify whether the scheme ensures message secrecy and anonymity of sender and receiver as it is assumed in the Huszti-Pethő protocol.

Model choices

The equational theory in Table 3.2 models the cryptographic primitives for reusable anonymous return channel. It includes models for ElGamal public-key encryption, digital signatures, and zero-knowledge proof. ElGamal encryption

consists of two functions *encryption* and *decryption*. A message encrypted with a public key can only be decrypted using the corresponding secret key. Digital signature consists of two equations: the function *getmess* checks integrity as it returns a message embedded into a signature; the function *checksign* checks also for authenticity as it returns the message only if the function is provided with the correct verification key. The theory for zero-knowledge proof that we use is inspired by Backes *et al.* [BMU08], who model zero-knowledge proof as two functions. The function *zkp_proof* models the proof that the prover builds to demonstrate the knowledge of the secret, which in the case of reusable anonymous return channel is the message encrypted with the public key of the mixer. The function *zkpsec* models the verification of the proof by the verifier, which in the case of reusable anonymous return channel is the mixer. The function $zkp\_proof(public, secret)$ takes as arguments public and secret parameters. In this case, the public parameter is the encryption of the message, and the private parameter is the message. Note that the correct function can be constructed only by the prover who knows the private parameter. The verification function $zkpsec(zkp\_proof(public, secret), verinfo)$ takes as arguments the proof function and the verification parameter *verinfo*. The verifier only accepts the proof if the relation between *verinfo* and *secret* is satisfied.

The ProVerif description of sender, receivers, and mixer processes is outlined in Figure 3.6. We specify one sender and two receivers, and model a simpler version of reusable anonymous return channel without considering the submission of reply, namely we omit step 3 of Figure 3.4. As we shall see later, this simpler version is sufficient to show successful attacks on message secrecy and sender anonymity.

An instance of reusable anonymous return channel is in Figure 3.7. We recall that submission of a message consists of the triplet $(\{ID_A, PK_A\}_{PK_M}, \{m\}_{PK_M}, \{ID_B, PK_B\}_{PK_M})$. We use the `choice` command in ProVerif to check both message secrecy and sender anonymity. This command allows us to verify if the processes obtained by instantiating a variable with two different values are bisimilar.

We analyse secrecy of the message by checking whether the attacker can distinguish the two scenarios in which the sender outputs two different messages. Thus, `choice` is applied in the second element of the triplet, and the process of the sender becomes as in Figure 3.8.

We analyse anonymity of the sender by checking whether the attacker can say if the message is sent either to Receiver 1 or Receiver 2. In this case, `choice` is applied in the first element of the triplet. Figure 3.9 shows the process of the sender to check anonymity.

Results

The results of the automatic analysis in ProVerif indicate that reusable anonymous return channel fails to guarantee both secrecy of messages and anonymity of sender and receiver identities. According the attack traces generated by ProVerif, both message secrecy and sender anonymity can be exploited using the same attack strategy. The attacker can use the mixer as decryption oracle, letting the mixer reveal any of the plaintexts contained in the triplet. The zero-knowledge proofs required to avoid this very attack reveal to be insufficient. In fact, the attack traces provided by ProVerif show the attacker can input the

```
(*----Sender----*)
let A (SKa: skey, PKb: pkey, PKmix: pkey, SPKmix: spkey) =
 out(c, (encrypt(pkey_to_bitstring(pk(SKa)), PKmix),
         encrypt(secret_message, PKmix),
         encrypt(pkey_to_bitstring(PKb), PKmix))).

(*----Receiver 1----*)
let B (SKb: skey, PKmix: pkey, SPKmix: spkey) =
 in (c, (c1: bitstring , s1: bitstring, cm1: bitstring)).

(*----Receiver 2----------------------*)
let C (SKc: skey, PKmix: pkey, SPKmix: spkey) =
 in (c, (c1: bitstring , s1: bitstring, cm1: bitstring)).

(*----Mixer----*)
let MIX (SKmix: skey, SSKmix: sskey ) =
 !MIX1(SKmix, SSKmix) | !MIX2(SKmix, SSKmix).

(*----Mixer 1----*)
let MIX1 (SKmix: skey, SSKmix: sskey ) =
 in (c, (c1: bitstring, c2: bitstring, c3: bitstring,
         p1:zkp, p2:zkp));
 let (xmsg: bitstring) = decrypt(c2, SKmix) in
 let (xdst: pkey) = bitstring_to_pkey(decrypt(c3, SKmix)) in
 if(checkproof(p1,c1) && checkproof(p2,c3)) then
  (out(c, (c1, sign(c1, SSKmix), encrypt( xmsg, xdst)))).

(*----Mixer 2----*)
let MIX2 (SKmix: skey, SSKmix: sskey ) =
 in (c, (c1': bitstring, c2': bitstring, c3': bitstring,
         p1':zkp, p2':bitstring));
 let (xmsg': bitstring) = decrypt(c2', SKmix) in
 let (xdst': pkey) = bitstring_to_pkey(decrypt(c3', SKmix)) in
 if(checkproof(p1',c1') && checksign(p2',spk(SSKmix)) = c3') then
  (out(c, (c1', sign(c1', SSKmix), encrypt( xmsg', xdst')))).
```

Figure 3.6: The processes of sender, receivers, and mixer.

mixer with valid zero-knowledge proofs.

In the following we detail the attack traces. The attacker chooses one of the three elements of the triplet. This choice depends on what the attacker wants to learn: if the target is the content of the message, the attacker chooses the second element; if the target is the identity of the sender, the attacker chooses the first element; if the target is the identity of the receiver, the attacker chooses the third element. Whatever the element of the triplet, the attacker submits this as a new message.

Figure 3.10 shows how the attacker can defeat sender anonymity. The at-

```
process
 new skA: skey; let pkA = pk(skA) in out (c, pkA);
 new skB: skey; let pkB = pk(skB) in out (c, pkB);
 new skC: skey; let pkC = pk(skC) in out (c, pkC);
 new skMIX: skey; let pkMIX = pk(skMIX) in out (c, pkMIX);
 new sskMIX: sskey; let spkMIX = spk(sskMIX) in out (c, spkMIX);
 (
        (A(skA, pkB, pkMIX, spkMIX)) |
        (B(skB, pkMIX, spkMIX)) |
        (C(skC, pkMIX, spkMIX)) |
        (MIX(skMIX, sskMIX))
 )
```

Figure 3.7: The instance of sender, receiver, and mixer processes.

```
(*----Sender----*)
let A (SKa: skey, PKb: pkey, PKmix: pkey, SPKmix: spkey) =
 out(c, (encrypt(pkey_to_bitstring(pk(SKa)), PKmix),
         encrypt(choice[secret_message1, secret_message2], PKmix),
         encrypt(pkey_to_bitstring(PKb), PKmix))).
```

Figure 3.8: The instance of sender to analyse message secrecy.

```
(*----Sender----*)
let A (SKa: skey, SKb:skey, PKb: pkey, PKc: pkey,
       PKmix: pkey, SPKmix: spkey) =
 out(c, (encrypt(pkey_to_bitstring(pk(choice[SKa, SKb]), PKmix),
         encrypt(secret_message, PKmix),
         encrypt(pkey_to_bitstring(PKb), PKmix))).
```

Figure 3.9: The instance of sender to analyse anonymity.

1. $A \rightarrow M$: $\{ID_A, PK_A\}_{PK_M}, \{m\}_{PK_M}, \{ID_B, PK_B\}_{PK_M}$
2. $I \rightarrow M$: $\{ID_A, PK_A\}_{PK_M}, \{ID_A, PK_A\}_{PK_M}, \{ID_I, PK_I\}_{PK_M}$
3. $M \rightarrow I$: $Sign_{SSK_M}(\{ID_A, PK_A\}_{PK_M}), \{ID_A, PK_A\}_{PK_I}$

Figure 3.10: Attack trace on sender anonymity

Primitive	Equation
ElGamal encryption	$decrypt(encrypt(m, pk(sk), r), sk) = m$
Digital signature	$getmess(sign(m, ssk)) = m$ $checksign(sign(m, ssk), spk(ssk)) = m$
Zero-knowledge proof	$zkpsec(zkp\_proof(encrypt(m, pk(sk), r), (r, m)),$ $encrypt(m, pk(sk), r)) = true$

Table 3.2: Equational theory to model reusable anonymous return channel

tacker targets $\{ID_A, PK_A\}_{PK_M}$, which becomes the second element of the new triplet submitted by the attacker. Note that the attacker can leave the first element of the triplet and the zero-knowledge proof unchanged. The attacker replaces the third element of the triplet with a public key PK_I for which the attacker knows the corresponding secret key SK_I. Thus, the attacker can also provide the necessary proof of knowledge of the plaintext contained in the third element. The mixer then shuffles the input messages, and encrypts the message with the attacker public key. Since the attacker knows the secret key SK_I, he can decrypt the message, which in this case is ID_A, PK_A, namely the identity of the sender.

Since the attacker can substitute any of the elements of the triplet as a new message, reusable anonymous return channel can neither ensure secrecy of the messages nor anonymity of sender and receiver. It can be observed that the checksum meant to guarantee the integrity of the triplet is added after the submission of the triplet, and is only used inside the mixer. Hence, the checksum does not prevent the attacker from submitting a modified triplet. Unfortunately even adding the checksum before the submission of the triplet does not prevent the attack as the knowledge of the ciphertexts is sufficient to compute the checksum.

Remark. Reusable anonymous return channel was originally designed to withstand a passive attacker that however can statically corrupt parties [GJ03]. Our analysis in ProVerif considers an active attacker. We observe that a passive attacker is not realistic in exam, where corrupted principals could actively try to cheat. However, an attacker who statically corrupts principals can still defeat reusable anonymous return channel. A corrupted principal can be instructed to send and receive messages via the reusable anonymous return channel on behalf of the attacker. The attacker still need to intercept those messages before they enter the mixer, but this is possible with insecure networks such as the Internet.

3.5.3 Formal Analysis of the Huszti-Pethő Protocol

We now introduce the formal model of the Huszti-Pethő protocol and then the results of the analysis of authentication and privacy.

Primitive	Equation
ElGamal encryption	$decrypt(encrypt(m, pk(sk), r), sk) = m$
Digital signature	$getmess(sign(m, ssk)) = m$ $checksign(sign(m, ssk), spk(ssk)) = m$
Zero-knowledge proof	$zkpsec(zkp\_proof(exp(exp(g, e1), e2), e2),$ $exp(exp(g, e1), e2)) = true$
Diffie-Hellman exp.	$exp(exp(exp(g, x), y), z) = exp(exp(exp(g, y), z), x)$
ZKP of discrete log.	$checkproof(xproof(p, p', t, exp(t, e), e),$ $p, p', t, exp(t, e)) = true$

Table 3.3: Equational theory to model the Huszti-Pethő protocol

Model choices

The first model choice is about the channels. The Huszti-Pethő protocol assumes that all messages are exchanged using reusable anonymous return channel. In the previous section, we demonstrated that reusable anonymous return channel fails to guarantee both message secrecy and sender anonymity. We choose to model the Huszti-Pethő protocol with the *ideal* implementation of reusable anonymous return channel, which guarantees anonymity of senders and receivers. This can be implemented with ProVerif's anonymous channels.

The equational theory depicted in Table 3.3 models the cryptographic primitives used within the Huszti-Pethő protocol. It includes the same models of ElGamal encryption, digital signatures, and zero-knowledge proofs defined for reusable anonymous return channels. In addition, we provide an equation to model the Diffie-Hellmann exponentiation. This model is limited because it just takes into account the equation needed for the protocol to work, and does not capture the full set of algebraic properties of Diffie-Hellmann exponentiation that an attacker may exploit to break the protocol. However, this has a limited influence on our analysis because, as we shall see later, the protocol ensures only one out ten security requirements, namely in most cases the attacker breaks the protocol though the simple model of Diffie-Hellmann exponentiation.

The equations for zero-knowledge proofs are customised according to the exponentiation operator. In particular, we support the model of zero-knowledge proof of the equality of discrete logarithms *check_proof* with tables in ProVerif. This approach is needed because ProVerif cannot deal with the associativity of multiple exponents. This approach is sound because it limits the attacker capability to generate fake zero-knowledge proofs, since the attacker cannot write and read ProVerif tables.

We assume that the same generator g is used to generate the pseudonyms of candidates and examiners. This choice is sound because we distinguish the roles, and each principal is identified by its public key. We replace the candidate identity with the corresponding pseudonym inside the events to check authentication requirements. Note that the replacement is also sound because the equational theory preserves the bijective mapping between the keys that identify the candidates and the pseudonyms.

The process of the exam authority is modelled as in Figure 3.11 and 3.12. It is conveniently split into four sub processes. Three sub processes concern prepa-

```
let EAi (SSKea: sskey,  SPKcom: spkey) =
(*Preparation*)
 new s: exponent;
 let h=exp(g, s) in
 let sh=sign(h,SSKea) in
 out(c, (h,sh));
 in(c, (quest: bitstring, squest: bitstring));
 (!Ereg(h,s) | !Creg(h,s,quest,squest, SPKcom, SSKea)).

let Ereg (h: bitstring, s: exponent)=
 (* Examiner Registration *)
 get keys (=g_e, xpk_e) in
 let q_tilde=exp(xpk_e, s) in
 out(c, (q_tilde, xpk_e));
 insert zkpeq(q_tilde, xpk_e);
 (* EA inserts q_tilde into a table to support the zkp with E *)
 in(c, (q: bitstring, q': bitstring));
 out(c, xproof(q,q',g, h, s));
 in (c, (t: bitstring, =q, =q'));
 if exp(q,s)=q' then
  in(c, zkp_sec_proof: bitstring);
  if zkpsec2(zkp_sec_proof,t, q)=true then 0.

let Creg (h: bitstring, s: exponent, quest: bitstring,
          squest: bitstring,  SPKcom: spkey, sskea:sskey)=
 (* Candidate Registration *)
 get keys (=g_c, xpk_c) in
 let p_tilde=exp(xpk_c, s) in
 insert reg_cand(xpk_c, p_tilde, h);
 out(c, (p_tilde, g_c));
 (* EA inserts p_tilde into a table to support the zkp with C *)
 insert zkpeq(p_tilde, xpk_c);
 in(c, (p: bitstring, p': bitstring));
 (* EA registered C with pseudonym 'p' to the exam 'h' *)
 out(c, xproof(p,p',g, h, s));
 Exam(h,s,quest,squest, SPKcom,sskea).
```

Figure 3.11: The process of the exam authority that concerns preparation.

```
let Exam (h: bitstring, s: exponent, quest: bitstring,
          squest: bitstring, SPKcom: spkey, SSKea: sskey)=
(*Testing*)
 in (c, (r: bitstring,p: bitstring,p':bitstring));
 if exp(p,s)=p' then
  in (c, zkp_sec_proof: bitstring);
  if zkpsec2(zkp_sec_proof, r, p)=true then
   if quest=checksign(squest, SPKcom) then
    out(c, (quest, squest));
    in (c, (=r, =p, answer: bitstring));
    (* EA succesfully collects the pair ('quest','answer') *)
    (* from C with pseudonym 'p' for the exam 'h' *)
    event collected(p', h, quest, answer);
    out(c,  hash( (r, p, p', zkp_sec_proof, quest, answer)));

(*Marking*)
    (* EA chooses an E who registered for the exam 'h' *)
    get examinertable(t, q, =h, xzkp_sec_proof) in
    new eap: bitstring;
    (* EA assigns 'eap' to the pair ('quest','answer') *)
    (* submitted by C with pseudonym 'p' for the exam 'h',*)
    (* and distributes them to E with pseudonym 'q' *)
    event distributed(p',h, quest, answer, eap, q);
    out(c, (eap, answer));
    get zkpeq(hma,=spkeytobitstring(spk(SSKea))) in
    in (c, (mark: bitstring, =hma, hma_e_enc: bitstring,
    zkp_sec_hma: bitstring, =t, =q));
    if (checkproof(zkp_sec_hma, hma, hma_e_enc,t,q)=true) then
     out(c, p');
(*Notification*)
     in (c, (=p', p_tilde: bitstring));
     get reg_cand(xpk_c, =p_tilde, =h) in
     insert marks(xpk_c,mark, hma, hma_e_enc, zkp_sec_hma, t,q).
```

Figure 3.12: The process of the exam authority that concerns testing, marking, and notification.

Requirement	Result	Time
Candidate Authorisation	✓	26 s
Answer Authenticity	×	3 s
Test Origin Authentication	×	3 s
Test Authenticity	×	33 s
Mark Authenticity	×	52 s
Question Indistinguishability	×	< 1 s
Anonymous Marking	×	1h 58 m 33 s
Anonymous Examiner	×	6h 37 m 33 s
Mark Privacy	×	23 m 59 s
Mark Anonymity	×	49 m 5 s

Table 3.4: Summary of the analysis of the Huszti-Pethő protocol.

ration and consist of the initialisation of the exam authority, the registration of the candidate, the registration of the examiner. The last sub process concerns the remaining phases, namely testing, marking, and notification.

The ProVerif model of the candidate is depicted in Figure 3.13, the examiner process is in Figure 3.14, the NET process is in Figure 3.15, and the exam process is modelled as in Figure 3.16. All the processes are augmented with the events that allow verifying the authentication requirements. To verify Question Indistinguishability we use the `noninterf` command of ProVerif, which checks that any two instances of the exam protocol that only differ in the value of the variable of questions are bisimilar. To verify Mark Privacy, Anonymous Examiner and Anonymous Marking we use the ProVerif command `choice[]`.

The full ProVerif code used to analyse the requirements of the Huszti-Pethő protocol is available on the Internet [Giu15].

Results

The analysis in ProVerif shows that the Huszti-Pethő protocol only ensures Candidate Authorisation as reported in Table 3.4. Regarding Answer Authenticity, ProVerif shows an attack trace that allows the exam authority to accept a test that has not been submitted by a registered candidate. In fact, the attacker can generate a fake pseudonym that allows him to take part in an exam for which the attacker did not register. This is possible because the exam authority does not check whether the pseudonym has been actually created using the partial information provided by the NET. The attacker generates a secret key SK_I, and calculates an associate pseudonym, which sends to the exam authority. The exam authority successfully verifies the received data since the attacker knows SK_I, hence the exam authority accepts the test. In other words, the exam authority may collect a test whose pseudonym is replaced with one chosen by the attacker. The same attack trace violates Test Origin Authentication because the attacker can generate a valid pseudonym for a candidate who did not register for the exam.

ProVerif finds a counterexample that invalidates Test Authenticity. The requirement cannot be achieved because there is no mechanism that allows the examiner to check whether the answers have been forwarded by the exam

```
let C (SKc: skey, SPKea: spkey, SPKcom: spkey) =
(*Preparation*)
 in (c, (h: bitstring, sh: bitstring));
 if h=checksign(sh, SPKea) then
  (* The zkp of equivalence of discrete log is supported by *)
  (* the tables 'zkpeq' and 'zkpeqnet'. *)
  (* In doing so, C can verify that 'ea_tilde' and 'p_tilde' *)
  (* have been correctly generated resp. by EA and NET *)
  get zkpeq(ea_tilde, =exp(g,skey_to_exponent(SKc))) in
  get zkpeqnet(=ea_tilde,r,p') in
  let p=exp(r,skey_to_exponent(SKc)) in
  out(c, (p, p'));
  in (c, zproof: bitstring);
  if (checkproof(zproof, p, p',g,h)=true) then
   let zkp_sec_c = zkp_proof2(r,p,skey_to_exponent(SKc)) in
   out(c, (r,p,p'));
   out(c, zkp_sec_c);

(*Testing*)
   in (c, (quest: bitstring, squest: bitstring));
   if quest=checksign(squest, SPKcom) then
    new answ: bitstring;
    (* C submits 'answ' and 'quest' for the exam 'h' *)
    event submitted(p', h, quest, answ);
    out(c, (r, p, answ));
    in (c, receipt: bitstring);
    if (hash( (r,p,p', zkp_sec_c, quest, answ))=receipt) then

(*Notification*)
     get marks(=exp(g,skey_to_exponent(SKc)),mark, hma, hma_e_enc,
     zkp_sec_hma, t,q) in
     get zkpeq(=hma,=spkeytobitstring(SPKea)) in
     if (hash((mark, answ)) = hma && checkproof(zkp_sec_hma, hma,
      hma_e_enc,t,q)=true ) then
      (* C is notified with 'mark' for the exam 'h' *)
      event notified(p',h,mark).
```

Figure 3.13: The process of the candidate.

```
let E (SKe: skey,  SPKea: spkey, SPKcom: spkey) =
 (*Preparation*)
 in (c, (h: bitstring, sh: bitstring));
 if h=checksign(sh, SPKea) then
  new alpha: exponent;
  get zkpeq(q_tilde, =exp(g,skey_to_exponent(SKe))) in
  let q'=exp(q_tilde, alpha) in
  let t=exp(g, alpha) in
  let q=exp(t, skey_to_exponent(SKe)) in
  out (c, (q, q'));
  in (c, zproof: bitstring);
  if (checkproof(zproof, q, q',g,h)=true) then
   let zkp_sec_e = zkp_proof2(t,q,skey_to_exponent(SKe)) in
   out(c, (t,q,q'));
   out(c, zkp_sec_e);
   insert examinertable (t, q, h, zkp_sec_e);

   (*Marking*)
   in (c, (quest: bitstring, squest: bitstring));
   if quest=checksign(squest, SPKcom) then
    in (c, (eap: bitstring, answer: bitstring));
    new mark: bitstring;
    event marked(quest, answer, mark, eap, q, h);
    let hma=hash( (mark, answer) ) in
    let hma_e= exp(hma, skey_to_exponent(SKe)) in
    insert zkpeq(hma, spkeytobitstring(SPKea));
    let zkp_sec_hma=xproof(hma,hma_e,t,q,skey_to_exponent(SKe)) in
    out(c, (mark, hma, hma_e, zkp_sec_hma, t, q)).
```

Figure 3.14: The process of the examiner.

```
let NET () =
(*Preparation*)
 in (c, (p_tilde: bitstring, =g_c));
 new ro: exponent;
 let p'=exp(p_tilde, ro) in
 let r=exp(g, ro) in
 (* The NET registered the candidate with pseudonym p' *)
 event registered(p');
 out(c,(p', r));
 (* The NET inserts 'p_tilde' into a table to support *)
 (* the zkp between EA (the prover) and C (the verifier) *)
 insert zkpeqnet(p_tilde,r,p');

(*Notification*)
 in (c, =p');
 out(c, (p', p_tilde)).
```

Figure 3.15: The process of the NET.

```
process
 !(
 new sskEA: sskey; let spkEA = spk(sskEA) in out (c, spkEA);
 new sskCom: sskey; let spkCom = spk(sskCom) in out (c, spkCom);
 new question: bitstring;
 let squestion=sign(question, sskCom) in
 out(c, (question, squestion));

 !(new skC: skey; let pkC = exp(g,skey_to_exponent(skC)) in
   out (c, pkC); insert keys(g_c, pkC); C(skC, spkEA, spkCom)
  )|
 !(EAi(sskEA , spkCom))|
 !(new skE: skey; let pkE = exp(g,skey_to_exponent(skE))
   in out (c, pkE); insert keys(g_e, pkE); E(skE,  spkEA, spkCom)
  )|
 !(NET())
 )
```

Figure 3.16: The exam process.

Requirement	Result	Time
Candidate Authorisation	✓	3 s
Answer Authenticity	×	2 s
Test Origin Authentication	✓	2 s
Test Authenticity	✓	3 s
Mark Authenticity	✓	4 s

Table 3.5: Summary of the analysis of authentication of the modified Huszti-Pethő protocol.

authority. Although the answer is encrypted with the public key of the mixer, this does not guarantee that the exam authority actually sent the message, because anyone can submit any message to the mixer.

Regarding Mark Authenticity, ProVerif provides an attack trace in which the attacker can forward any answer to any examiner, even if the answer was not collected by the exam authority. Moreover, no mechanism ensure that the notified mark originates from the exam authority. In fact, the attacker can notify the candidate by himself with a mark of his choice.

The Huszti-Pethő exam protocol does not guarantee any privacy requirement. Intuitively, all privacy requirements can be violated because reusable anonymous return channel does not guarantee anonymity.

However, even assuming anonymous channels, ProVerif shows an attack trace for each requirement. Question Indistinguishability does not hold because messages sent via reusable anonymous return channel are not secret, as our analysis demonstrates. Since the questions are sent through the anonymous channel, the attacker can still obtain them.

Anonymous Marking is violated since the attacker can check whether a candidate accepts the zero-knowledge proof, hence associates the candidate identity with the pseudonym, and then identifies the candidate's test.

Anonymous Examiner can be also violated because the attacker can track which examiner accepts the zero-knowledge proof when receiving the partial pseudonym, hence associates the answer to the examiner.

Mark Privacy fails because the examiner sends the mark to the exam authority via reusable anonymous return channel, which does not ensure secrecy.

Finally, ProVerif shows that the Huszti-Pethő protocol does not also ensure *Mark Anonymity*. Since one can track which pseudonym is assigned to the candidate, and the mark is not secret, the attacker can link the candidate to the assigned mark.

3.5.4 Fixing Authentication

We propose four modifications to the Huszti-Pethő protocol in order to achieve most of authentication requirement. In particular, we prove in ProVerif that the modified Huszti-Pethő protocol achieves Candidate Authorisation, Test Origin Authentication, Test Authenticity, and Mark Authenticity as in Table 3.5. We found no easy solution for Answer Authenticity because the protocol sees no signatures for candidates, and reusable anonymous return channel does not guarantee authentication. The first modification consists in the NET receiving

the partial pseudonyms generated by the exam authority via a secure channel instead via reusable anonymous return channel. It can be observed that the exam authority and the NET do not need to communicate anonymously via RARC, as the original protocol prescribes. Conversely, they need a secure channel to avoid that the attacker injects messages. In doing so, the attacker cannot use the NET to generate fake pseudonyms.

Similarly, the second modification consists in the exam authority receiving the eligible pseudonyms from NET via a secure channel. Thus, the exam authority generates zero-knowledge proofs of the equality of discrete logarithm to eligible pseudonyms only. The exam authority can also store the eligible pseudonyms, which can be checked at testing before the exam authority accepts a test from a candidate.

The third modification concerns marking and consists in the exam authority signing the collected test prior to distribute it to the chosen examiner. In fact, it is required the examiner identity to be anonymous but not the exam authority's. Thus, the examiner marks the test only if the signature can be correctly verified. In the original protocol, the examiner could not verify whether a test was sent by the exam authority.

The last modification concerns the test identifier the exam authority affixes to the test before distributing it to the examiner, and consists in a modified receipt of candidate's submission. The exam authority adds the test identifier to the receipt and signs it. The examiner also adds the test identifier into the receipt of marking, hence the candidates can verify whether they are notified with the correct marks. In the original protocol, the attacker could notify the candidate with any other examiner's mark because the candidate was unaware of the test identifier.

Table 3.4 summarises the results of the formal analysis of the Huszti-Pethő protocol assuming all principal being honest. The reported times refer to ProVerif analyses over an Intel Core i7 3.0 GHz machine with 8 GB RAM.

It can be seen that the modified Huszti-Pethő protocol guarantees four out five authentication requirements. Unfortunately, no easy solution can be envisaged for privacy. The design of the original protocol is heavily based on the assumption that reusable anonymous return channel guarantees secrecy, authentication, and anonymity. Since reusable anonymous return channel guarantees none of these properties, the Huszti-Pethő protocol would need a complete redesign to achieve privacy.

3.6 Conclusion

This chapter discusses a formal framework for the security analysis of exam protocols. Although many symbolic analysis methods have been proposed, our choice to model exams in the applied π-calculus reveals to be interesting. We advance five authentication and five privacy requirements for exam, counting a total of ten novel requirements. The proposed framework is validated with the formal analysis of the Huszti-Pethő protocol, the first secure exam scheme proposed in the literature. The protocol succumbed to our analysis, though being quite complex.

It is found that the protocol guarantees only one of the ten requirements. Authentication is compromised because of inaccuracies in the protocol design. Pri-

vacy requirements are mostly violated because assumptions on reusable anonymous return channel. It is demonstrated that an attacker can compromise message secrecy and anonymity on reusable anonymous return channel.

This chapter also introduces a few modifications on the Huszti-Pethő protocol in order to guarantee most of the authentication requirements. A formal analysis in ProVerif confirms that the modified protocol ensures these requirements. However, even with an ideal reusable anonymous return channel implementation that ensures anonymity, the Huszti-Pethő protocol does not ensure any privacy requirements. Thus, we think that the protocol requires fundamental changes.

Generally speaking, the proposed formal framework brings exams into the attention of the security community. Computer-based exams are becoming widespread, and it can be difficult to discover exam protocol vulnerabilities as they may be exposed to unprecedented cheating attacks.

This work poses the first research step in the formal understanding of exam protocols. This is corroborated by the analysis of more protocols as discussed in chapters 5 and 6.

Chapter 4

Formalising Verifiability

A fundamental requirement of exams is transparency. While traditional exams can be normally observed through all the phases of the exam, computer-assisted exams may introduce opaqueness. In general, any operation performed by computers may not be observable, depending on the level of computer assistance. For example, computer markers may alter the evaluations of the tests, or a malware in the collector may modify or drop the submitted tests. Thus, transparency demands for *verifiable* exams. A verifiable exam can be checked for the presence or the absence of irregularities and provides evidences about fairness and correctness of marking. Moreover, verifiable exams foster public trust, as transparency can persuade the involved parties to comply with regulations.

To analyse whether an exam protocol is verifiable, it is necessary to clarify the relevant verifiability requirements, and then build a framework to check the protocol against these requirements. In this chapter, we present a clear understanding of verifiability for exam protocols and propose a methodology to analyse their verifiability. Differently from the framework advanced in chapter 3, we propose a formal framework based on multisets that abstract away from the applied π-calculus constraints and is suitable for both symbolic and computational analysis. We also formalise eleven verifiability requirements for exams. Each requirement is pivoted on a *verifiability-test*, and we state the conditions that a sound and complete verifiability-test has to satisfy. Following a practice already explored in other domains [CF85, BT94, Ben96, HS00], we classify our verifiability requirements into individual and universal.

Outline of the chapter. Section 4.1 discusses the related work about the formalisation of verifiability in other contexts. Section 4.2 introduces the constituents of the formal framework. Section 4.3 contains the specification of eleven verifiability requirements for exams. Section 4.4 draws the conclusions and outlines the future work.

4.1 Related Work

To the best of our knowledge, verifiability for exams has not been studied at the time of writing this manuscript. Few papers list informally a few security requirements. Castella-Roca *et al.* [CRHJDJ06] discuss a secure exam man-

agement system, and informally define authentication, privacy, correction and receipt fullness. Huszti and Pethő [HP10], whose protocol is analysed in chapter 3, extend the requirements with secrecy and robustness. None of the works outlined above addresses verifiability.

However, verifiability has been studied in domains close to exams, such as voting and auctions, and different models and requirements have been proposed from the beginning of 2010s [DHL13, KuTV10, KRS10]. In voting, *individual verifiability* signifies that voters can verify that their votes have been handled correctly, namely "cast as intended", "recorded as cast", and "counted as recorded" [BT94, HS00]. The concept of *universal verifiability* has been introduced to express that auditors can verify the correctness of the tally using only public information [CF85, BT94, Ben96]. Kremer *et al.* [KRS10] formalised both individual and universal verifiability in the applied π-Calculus. They also introduced the requirement of *eligibility verifiability*, which expresses that auditors can verify that each vote in the election result was cast by a registered voter, and there is at most one vote per voter. Smyth *et al.* [SRKM10] used ProVerif to check verifiability in three voting protocols. They express the requirements as reachability properties. In the next chapter, we also analyse an exam protocol in ProVerif to validate our framework. However, our model and definitions are constrained neither to the applied π-Calculus nor to ProVerif. The sound and complete verifiability-tests that we use to specify our requirements are inspired by the work of Dreier *et al.* [DHL13], who formalised verifiability for e-auction.

Küsters *et al.* [KuTV10] studied *accountability*. This requirement says that, in presence of a protocol failure, one can identify the principal responsible for the failure. The notion of accountability is strongly related to verifiability as the latter's goal is to check the presence of protocol failures. In their work, Küsters *et al.* give symbolic and computational definitions of verifiability, which they argue to be a weak variant of accountability. Differently from our approach, their framework has to be instantiated for each application by identifying relevant verifiability goals.

Guts *et al.* [GFZN09] defined *auditability* as the quality of a protocol, which stores a sufficient number of evidences, to convince an honest judge that specific properties are satisfied. As we shall see later, auditability expresses the same concept of universal verifiability as defined in this dissertation: anyone, even an outsider without a private knowledge about the protocol execution, can verify that the system relies only on the available pieces of evidence.

4.2 A More Abstract Model

In chapter 2, we observed that any exam involves at least the candidate role plus other possible authority roles, and that the run of an exam can be represented as the sequential execution of preparation, testing, marking, and notification phases. From these observations, we build a formal framework for the analysis of verifiability requirements. While in chapter 3 we advanced a framework based on the applied π-calculus, hence supported by an operational semantics, here we propose an abstract framework for the analysis of verifiability that comes without such semantics. On the one hand, a more abstract framework requires to map the model onto one that allows the analysis of the protocol. On the other hand, it allows for a wider choice of analysis methods, regardless whether they

are based on the symbolic or computational model. In chapter 5 we validate this framework by analysing verifiability in a novel exam protocol, using ProVerif as analysis method.

We view the abstract model of an exam consisting of four sets, three relations, and one function. The four sets are of candidate's identities I, questions Q, answers A, and marks M. The three relations `Accepted`, `Marked`, and `Assigned`, link candidates, questions, answers, and marks along the four phases. A test consists of the pair $(Q \times A)$ of questions and answers. The function `Correct` maps a mark to a test. It is assumed that sets and relations are built from data logs such as registers or repositories.

Definition 28 (Exam (abstract model)) *An exam E is a tuple (I, Q, A, M, α) where I of type $\mathcal{I}$ is a set of candidate identities, Q of type $\mathcal{Q}$ is a set of questions, A of type $\mathcal{A}$ is a set of answers, M of type $\mathcal{M}$ is a set of marks, and α is the set of the following relations:*

- `Accepted` $\subseteq I \times (Q \times A)$*: the candidates' tests accepted by the collector authority;*
- `Marked` $\subseteq I \times (Q \times A) \times M$*: the marks given to the candidates' tests;*
- `Assigned` $\subseteq I \times M$*: the marks assigned to the candidates;*
- `Correct` $: (\mathcal{Q} \times \mathcal{A}) \to \mathcal{M}$*: the function used to mark a test.*

Definition 28 can be extended with two specific subsets:

- $I_r \subseteq I$ as the set of identities of candidates who registered for the exam;
- $Q_g \subseteq Q$ as the set of questions generated by the question committee.

It can be noted that this approach can model exam executed either honestly or with frauds. For example, the set $I \setminus I_r$ contains the identities of the unregistered candidates who took the exam. Similarly, the set $Q \setminus Q_g$ contains the illegitimate questions administered at the exam. An honest execution of an exam requires $(I \setminus I_r) = (Q \setminus Q_g) = \emptyset$

The function `Correct` models any objective mapping that assigns a mark to an answer. This works well for multiple-choice tests, but it is inappropriate for free-response tests. The evaluation of a free-response question is hardly objective: the ambiguities of natural language can lead to subjective interpretations by the examiner. In this case, it is not possible to verify the correctness of the marking, whatever model is considered. In other words, an exam protocol that does not allow a definition of the function `Correct` cannot be checked for the correctness of the marking.

4.3 Verifiability Requirements

In this section we present eleven verifiability requirements for exams. Verifiability is a sort of meta-requirement as a protocol is verifiable with respect to specific properties.

To be verifiable an exam should be *testable*, namely it should provide an executable procedure (*verifiability-test*) that checks a specific property on the exam

execution. For executable procedure we refer to the existence of a procedure that takes in some data and outputs true or false. Such procedure may be either explicitly provided by the protocol designer or found by the protocol analyst. A verifiability requirement has the form $t(e) \Leftrightarrow c$, where the verifiability-test $t(e)$ is a function from $\mathcal{E} \rightarrow \texttt{bool}$, where $\mathcal{E}$ is the set of data e, and c is a predicate that models the specific property. The data needed to run the verifiability-test are obtained from the available information about the execution of the exam and from the private knowledge of the involved roles. It is assumed that the pieces of data become available after the exam concludes and are not subject to further changes. The following definition resumes the notion of testable exam.

Definition 29 (Testable exam) *An exam protocol is* testable *if it provides a verifiable-test that checks a desired property.*

Being testable is not a sufficient condition for an exam to be verifiable because the verifiability-test should be sound and complete ($\Leftrightarrow$) for the specific property: the success of the verifiability-test is a *sufficient* condition for c to hold (soundness $\Rightarrow$), and the success of the verifiability-test is a *necessary* condition for c to hold (completeness $\Leftarrow$). This is captured by the definition of *verifiable exam.*

Definition 30 (Verifiable exam) *An exam protocol is* verifiable *for a desired property if the exam is testable and the corresponding verifiable-test is* sound *and* complete*.*

With a security take, a verifiability-test should be sound in presence of an attacker and corrupted principals. It means that when the test succeeds the property holds despite the presence of attacker and corrupted principals. The verifiability-test should be complete to avoid trivialities: a verifiability-test that always returns false is sound but useless.

Sound verifiability-tests cannot be complete if a corrupted principal is allowed to submit incorrect data, since the verifiability-test would fail although the property holds. Thus, a verifiability-test should be complete in the sense that if all principals follow the protocol, then the verifiability-test must succeed.

A verifiability-test can be run by exam principals or outsiders, a distinction that leads to two notions of verifiability requirements: *individual* and *universal.* In the scenario of exam, we view individual verifiability as verifiability from the perspective of the candidate role. The candidate can feed the verifiability-test with the private knowledge acquired during the exam, namely the candidate's identity, the test, the mark, and the messages the candidate exchanged with the other principals through the exam.

We view universal verifiability as verifiability from the perspective of an external auditor or outsider. This role can be played by auditors who acquire no private knowledge during the exam. The auditor typically has no tasks associated to an exam, thus he has no candidate's identity, he has not seen the exam's questions, answered any of them, and he did not receive any mark. Besides, he has not interacted with any of the exam principals. In short, the auditor runs the verifiability-tests only using the exam's available pieces of data.

The list of proposed verifiability requirements is not meant to be exhaustive but aims to cover all the phases of an exam. The requirements concern the verifiability of candidate registration, the validity of questions, and the integrity

Requirement	Individual Verifiability	Universal Verifiability
Registration		$\mathtt{R}_{\mathtt{UV}}(e) \Leftrightarrow$ $I_r \supseteq \{i : (i,x) \in \mathtt{Accepted}\}$
Question Validity	$\mathtt{QV}_{\mathtt{IV}}(i,q,a,m,p) \Leftrightarrow (q \in Q_g)$	
Marking Correctness	$\mathtt{MC}_{\mathtt{IV}}(i,q,a,m,p) \Leftrightarrow$ $(\mathtt{Correct}(q,a) = m)$	$\mathtt{MC}_{\mathtt{UV}}(e) \Leftrightarrow$ $(\forall (i,x,m) \in \mathtt{Marked},$ $\mathtt{Correct}(x) = m$
Test Integrity	$\mathtt{ETI}_{\mathtt{IV}}(i,q,a,m,p) \Leftrightarrow$ $\big((i,(q,a)) \in \mathtt{Accepted}$ $\wedge \exists m' : (i,(q,a),m') \in \mathtt{Marked}\big)$	$\mathtt{ETI}_{\mathtt{UV}}(e) \Leftrightarrow \mathtt{Accepted} =$ $\{(i,x) : (i,x,m) \in \mathtt{Marked}\}$
Test Markedness	$\mathtt{ETM}_{\mathtt{IV}}(i,q,a,m,p) \Leftrightarrow$ $(\exists m' : (i,(q,a),m') \in \mathtt{Marked}))$	$\mathtt{ETM}_{\mathtt{UV}}(e) \Leftrightarrow \mathtt{Accepted} \supseteq$ $\{(i,x) : (i,x,m) \in \mathtt{Marked}\}$
Marking Integrity	$\mathtt{MI}_{\mathtt{IV}}(i,q,a,m,p) \Leftrightarrow$ $\exists m' : \big((i,(q,a),m') \in \mathtt{Marked}$ $\wedge (i,m') \in \mathtt{Assigned}\big)$	$\mathtt{MI}_{\mathtt{UV}}(e) \Leftrightarrow \mathtt{Assigned} =$ $\{(i,m) : (i,x,m) \in \mathtt{Marked}\}$
Marking Notification Integrity	$\mathtt{MNI}_{\mathtt{IV}}(i,q,a,m,p) \Leftrightarrow$ $(i,m) \in \mathtt{Assigned}$	

Table 4.1: Individual and Universal Verifiability

of tests, marks, and notification. In the remainder of the section we detail the requirements, which are concisely listed in Table 4.1. Generally speaking, an exam is fully verifiable, if it ensures all the verifiability requirements.

4.3.1 Individual Verifiability

Individual verifiability allows the candidate to verify some aspects of the exam using the public data that is available from the execution of the exam plus the candidate's private knowledge. The candidate knows her identity i, the test she submitted, which consists of question q and answer a, and the notified mark m. The candidate also knows the perspective p of the run of the exam. The perspective consists of the messages the candidate sent and received during the run of the exam. Thus, the data is a tuple (i,q,a,m,p). Note that the candidate's perspective p is not necessary to specify the predicate that models the properties to verify. In fact, the perspective never appears in the predicate c. However, the perspective may be necessary to implement the verifiability-test $t(e)$.

The six individual verifiability requirements concern the validity of the questions, the integrity of the submitted test, and the correctness and integrity of the mark notified to the candidate.

The first requirement is *Question Validity*, which signifies that the candidate can check that she received the questions actually generated by the question committee. The requirement is modelled by a verifiability-test that returns true if the questions q received by the candidate belong to the set of the valid questions Q_g generated by the question committee. This is formalised as follows:

Definition 31 (Question Validity I.V.) *Given an exam E and a set of verifiability-tests β, then (E,β) is* question validity verifiable *if there is a verifiability-test $\mathtt{QV}_{\mathtt{IV}} : \mathcal{E} \to \mathtt{bool}$ in β such that*

$$\mathtt{QV}_{\mathtt{IV}}(i,q,a,m,p) \Leftrightarrow (q \in Q_g)$$

The next requirement is *Marking Correctness*, which says that the candidate

can verify that the mark she received is correctly computed on her test. It can be formalised as:

Definition 32 (Marking Correctness I.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* marking correctness verifiable *if there is a verifiability-test* $\mathtt{MC}_{\mathtt{IV}} : \mathcal{E} \to \mathtt{bool}$ *in β such that*

$$\mathtt{MC}_{\mathtt{IV}}(i, q, a, m, p) \Leftrightarrow (\mathtt{Correct}(q, a) = m)$$

A way to ensure Marking Correctness is to give the candidate access to the marking algorithm, so she can compute again the mark and compare it with the one she received. As we discussed in Section 4.2, this makes perfect sense with multiple-choice tests, but it makes not in the case of free-response tests. However, one can envisage other ways to convince the candidate she received the correct mark, provided the examiner follows the marking algorithm correctly. For example, the candidate could check that the integrity of her test is preserved from submission until marking, and that the integrity of the mark is preserved from marking until notification. The remaining individual verifiability requirements cover these very checks.

The third requirement is *Test Integrity*, which states that the candidate can check that her test is accepted and marked as she submitted it. It is formalised as follows:

Definition 33 (Test Integrity I.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* test integrity verifiable *if there is a verifiability-test* $\mathtt{ETI}_{\mathtt{IV}} : \mathcal{E} \to \mathtt{bool}$ *in β such that*

$$\mathtt{ETI}_{\mathtt{IV}}(i, q, a, m, p) \Leftrightarrow \big((i, (q, a)) \in \mathtt{Accepted} \wedge \exists m' : (i, (q, a), m') \in \mathtt{Marked}\big)$$

Since the verifiability-tests are run after the conclusion of the exam, Test Integrity cannot capture the scenario in which a test is modified before the marking and put back to its original version after marking. Such scenario can be detected by verifying Marking Correctness.

Another requirement that concerns the integrity of the test is *Test Markedness*, which says that the candidate can check that the test she submitted is marked without modification. It can be specified as follows:

Definition 34 (Test Markedness I.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* test markedness verifiable *if there is a verifiability-test* $\mathtt{ETM}_{\mathtt{IV}} : \mathcal{E} \to \mathtt{bool}$ *in β such that*

$$\mathtt{ETM}_{\mathtt{IV}}(i, q, a, m, p) \Leftrightarrow (\exists m' : (i, (q, a), m') \in \mathtt{Marked})$$

Note that the predicate of Test Markedness coincides with the one of Test Integrity pruned of "$(i, (q, a)) \in \mathtt{Accepted}$". Thus, if $\mathtt{ETI}_{\mathtt{IV}}$ succeeds, then $\mathtt{ETM}_{\mathtt{IV}}$ also succeeds, namely $\mathtt{ETI}_{\mathtt{IV}}(i, q, a, m, p) \Rightarrow \mathtt{ETM}_{\mathtt{IV}}(i, q, a, m, p)$. However, if the $\mathtt{ETI}_{\mathtt{IV}}$ fails, but $\mathtt{ETM}_{\mathtt{IV}}$ succeeds, it follows that the test of the candidate is modified upon acceptance but put back to its original version before marking. This may be not a security issue for the candidate since her test is marked as submitted. However, the candidate can report this issue to the responsible authority for further investigation. Another scenario where an exam protocol

may provide $\texttt{ETM}_{\texttt{IV}}$ but not $\texttt{ETI}_{\texttt{IV}}$ is when there is a lack of available data at the conclusion of the exam.

The next requirement is *Mark Integrity*, which signifies that the candidate can verify that the mark attributed to her test is assigned to her without any modification. This requirement is formalised as follows:

Definition 35 (Mark Integrity I.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* mark integrity verifiable *if there is a verifiability-test $\texttt{MI}_{\texttt{IV}} : \mathcal{E} \to \texttt{bool}$ in β such that*

$$\texttt{MI}_{\texttt{IV}}(i, q, a, m, p) \Leftrightarrow \exists m' : \big((i, (q, a), m') \in \texttt{Marked} \wedge (i, m') \in \texttt{Assigned}\big)$$

The last requirement is *Mark Notification Integrity*, which says that the candidate can check she received the mark assigned to her. This requirement is formalised as:

Definition 36 (Mark Notification Integrity I.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* mark notification integrity verifiable *if there is a verifiability-test $\texttt{MNI}_{\texttt{IV}} : \mathcal{E} \to \texttt{bool}$ in β such that*

$$\texttt{MNI}_{\texttt{IV}}(i, q, a, m, p) \Leftrightarrow (i, m) \in \texttt{Assigned}$$

There is a subtle difference between the two last definitions. $\texttt{MI}_{\texttt{IV}}$ can succeed despite the candidate is notified with a mark that is different from the one assigned to her, while $\texttt{MNI}_{\texttt{IV}}$ cannot. Conversely, if $\texttt{MNI}_{\texttt{IV}}$ succeeds, then $\texttt{MI}_{\texttt{IV}}$ could fail if the examiner evaluated the test with a different mark.

4.3.2 Universal Verifiability

The definitions of universal verifiability are not pivoted around any exam role, but consider the viewpoint of an external auditor. The auditor runs the verifiability-tests on the public data available after an exam protocol run. Hence, the knowledge of the auditor consists of a general variable e that contains the data.

The five universal verifiability requirements concern the registration of the candidates and the integrity of the batch of tests from the submission until after the marking. The requirements of Question Validity and of Mark Notification Integrity, which are definitions relevant for individual verifiability, are hard to capture in the context of universal verifiability. This is because the external auditor has no knowledge of the questions nor of the marks received by the candidates, but only of public data.

The first universal verifiability requirement we consider is *Registration*, which says that an auditor can check that all accepted tests are submitted by registered candidates. Thus, the collector should have considered only tests that originated from eligible candidates. This requirement can be specified as:

Definition 37 (Registration U.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* registration verifiable *if there is a verifiability-test $\texttt{R}_{\texttt{UV}} : \mathcal{E} \to \texttt{bool}$ in β such that*

$$\texttt{R}_{\texttt{UV}}(e) \Leftrightarrow I_r \supseteq \{i : (i, x) \in \texttt{Accepted}\}$$

Note that the superset symbol is preferred over strict equality since a candidate may register for an exam but may not show at testing. Thus, the collector may accept fewer tests than registered candidates.

The next requirement is *Marking Correctness*, which signifies that an auditor can check that all the marks attributed by the examiners to the tests are computed correctly. It is formalised as follows:

Definition 38 (Marking Correctness U.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* marking correctness universally verifiable *if there is a verifiability-test* $\mathtt{MC}_{\mathtt{UV}} : \mathcal{E} \rightarrow \mathtt{bool}$ *in β such that*

$$(\mathtt{MC}_{\mathtt{UV}}(e)) \Leftrightarrow (\forall (i, x, m) \in \mathtt{Marked},\ \mathtt{Correct}(x) = m))$$

Marking Correctness makes the same arguments about free-response of tests we observed for Marking Correctness. However, for the sake of transparency, it can be assumed that exam authorities allow auditors to access the logs of the exam such that the auditors can inspect the marking process.

The third requirement is *Test Integrity*, which says that an auditor can verify that all and only accepted tests are marked without any modification. It means that the auditor can be convinced that no test is modified, added, or deleted until the end of marking. This requirement is formalised as:

Definition 39 (Test Integrity U.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* test integrity universally verifiable *if there is a verifiability-test* $\mathtt{ETI}_{\mathtt{UV}} : \mathcal{E} \rightarrow \mathtt{bool}$ *in β such that*

$$(\mathtt{ETI}_{\mathtt{UV}}(e)) \Leftrightarrow (\mathtt{Accepted} = \{(i, x) : (i, x, m) \in \mathtt{Marked}\})$$

The equality symbol in the predicate specification enforces that at marking no test has been added or removed from the batch of accepted tests.

The next requirement is *Test Markedness*, which says that an auditor can check that only the accepted tests are marked without modification. It is formalised as follows:

Definition 40 (Test Markedness U.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* test markedness universally verifiable *if there exists a verifiability-test* $\mathtt{ETM}_{\mathtt{UV}} : \mathcal{E} \rightarrow \mathtt{bool}$ *in β such that*

$$(\mathtt{ETM}_{\mathtt{UV}}(e)) \Leftrightarrow (\mathtt{Accepted} \supseteq \{(i, x) : (i, x, m) \in \mathtt{Marked}\})$$

It can be noted that Test Markedness is a relaxed version of Test Integrity because the predicate of the former definition does not require strict equality of the two multisets. Thus, if $\mathtt{ETI}_{\mathtt{UV}}$ fails but $\mathtt{ETM}_{\mathtt{UV}}$ succeeds, it follows that at least one accepted test has not been marked. This scenario however may not be a security problem. For example, the rules of the exam may allow the candidate to drop the examination after testing. Conversely, the scenario in which an examiner marks a test that was not accepted is normally considered a violation of the exam.

The last requirement we consider is Mark Integrity, which signifies that an auditor can check that all and only the marks associated to the tests are assigned to the corresponding candidates with no modifications. This is formalised as:

Definition 41 (Mark Integrity U.V.) *Given an exam E and a set of verifiability-tests β, then (E, β) is* mark integrity universally verifiable *if there exists a verifiability-test* $\mathtt{MI}_{\mathtt{UV}} : \mathcal{E} \rightarrow \mathtt{bool}$ *in β such that*

$$(\mathtt{MI}_{\mathtt{UV}}(e)) \Leftrightarrow (\mathtt{Assigned} = \{(i, m) : (i, x, m) \in \mathtt{Marked}\})$$

The equality symbol in the specification of the predicate enforces that no pair of candidates and marks have been added or removed from the batch of marked tests.

To conclude, it can be observed that the combination of registration, test integrity, and mark integrity universal verifiability enforces the verifiability from preparation to notification of an exam protocol.

4.4 Conclusion

This chapter advances verifiability requirements for exam protocols. The domain of exam has unique features that call for verifiability definitions that are different from ones proposed for voting and auctions. The eleven requirements, which are classified in individual and universal verifiability categories, are specified in a formal and abstract model that opens up opportunities for both symbolic and computational analysis. This model contrasts the frameworks proposed for the verifiability of voting and auction protocols, as the latter usually focus on cryptographic protocols. Intuitively, the proposed model is sufficiently abstract to specify any type of exam, from traditional to Internet-based exams. Traditional exams usually provide evidence data via log-books and registers, while Internet-based exams implement the electronic versions, such as web bulletin-boards.

Individual verifiability definitions consider a candidate who can check if she got a valid set of questions, if her test was properly processed through the phases of the exam, and if her mark was correctly computed. Universal verifiability definitions consider an external auditor who can check the correct execution of the exam with no private knowledge about the run of the exam. The auditor can verify if only registered candidates took the exam, if all tests were properly processed and marked, and if the marks were assigned correctly.

We validate the proposed model in the next chapter, in which we introduce an Internet-based exam protocol. We analyse it for the eleven verifiability requirements, and show how to map the relations defined in the proposed abstract model in ProVerif.

Chapter 5

The Remark! Internet-based Exam Protocol

In the last decade, computers have been extensively introduced in the design of critical systems such as voting, auctions, and exams. Computers are progressively becoming the main components of such systems providing the core tasks. In the context of exam, computers can be used for local tasks, such as generation of questions and automatic marking, but also to support remote tasks. For example, remote registration and remote notification of candidates are must-have functionalities of exams that expect many candidates; remote testing, in which distant located candidates take the exam at their place, is the distinctive functionality of Internet-based exams. At an extreme, all the phases of an exam may take place remotely.

The use of computers exposes exams to new threats and requires changing the well-established procedures used in traditional exam. The design of secure exam protocols is further complicated by the conflicting interests that roles typically have in an exam. In fact, it may be hard to find a role who can play as TTP, as recent exam scandals confirm [Cop13, Wat14, Lip14]. In this chapter, we advance Remark!, a new Internet-based exam protocol that guarantees several authentication, privacy, and verifiability requirements without the need of a TTP. The idea behind Remark! is to distribute the trust across the several servers that compose an *exponentiation mixnet*. As we shall see later, the mixnet generates the pseudonyms that allow the exam principals to encrypt and sign messages anonymously. Using ProVerif, we prove that Remark! ensures all the authentication and privacy requirements proposed in chapter 3 with minimal reliance on trusted parties. Moreover, we demonstrate that Remark! provides the verifiability-tests listed in chapter 4, and discuss the necessary assumptions to make Remark! fully verifiable.

Assumptions

Like any other security protocol, Remark! is not designed to withstand every possible threat. For instance, it cannot cope with plagiarism, but assumes ap-

propriate invigilation during testing. Principals may still collude and communicate via subliminal channels, for instance by using steganography. Although it is hard to rule out completely such a threat, steganalysis techniques can be of some help here. Other countermeasures may be needed against collusion attacks that exploit covert channels. We thus specify seven assumptions conveniently for the goals of Remark!. In particular, we assume that:

1. Each principal holds a long-term public/private pair of keys.
2. The candidate holds a smart card in which the personal details of the candidate are visibly engraved. The smart card securely stores the candidate's private key, namely the private key cannot be extracted from the smart card.
3. The candidate is invigilated during testing to mitigate cheating. Invigilation for remote testing can be guaranteed with online invigilation software, such as ProctorU [Inc15].
4. The model answers are kept secret from the candidates until after testing. The examiners may be provided with the model answers at marking.
5. It is available an authenticated append-only bulletin board that guarantees everyone to see the same data, though write access might be restricted to appropriate entities [BRT13]. An implementation of a bulletin board and its formal analysis has been proposed by Culnane and Schneider [CS14].
6. It is available an implementation of TLS channel that ensures integrity and confidentiality of messages.
7. At least one of the servers that compose the exponentiation mixnet is honest.

Outline of the chapter. Section 5.1 reviews a few proposals of secure protocols for Internet-based exams. Section 5.2 details exponentiation mixnet, a cryptographic scheme on which Remark! is based. Section 5.3 describes Remark! according the four phases of an exam. Section 5.4 contains the formal analysis in ProVerif of ten authentication and privacy requirements Section 5.5 contains the formal analysis in ProVerif of eleven verifiability requirements. Finally, Section 5.6 discusses future work and concludes the chapter.

5.1 Related work

To the best of our knowledge only few works propose protocols for Internet-based exams. TOEFL [TOE], which is one of the major English-language test in the world, has replaced its traditional exams with Internet-based exams. Neither the specification nor the security requirements of the TOEFL protocol are publicly available. Moreover, its design probably includes a trusted exam authority that is in charge of the critical tasks of the exam. The same concerns apply for MOOCs, which offer Internet-based exams that grant credits for many universities [Lew13]. Conversely, Remark! is designed to minimise the reliance on trusted parties. Huszti-Pethő [HP10] advanced an Internet-based exam with

few trust requirements on principals, but in chapter 3 we have shown that the protocol has several security issues. In contrast, we prove that Remark! ensure all the security requirements.

In the domain of Computer Supported Collaborative Working, Foley and Jacob [FJ95] formalised confidentiality requirements and proposed an exam as case study. Maffei *et al.* [MPR13] implemented a course evaluation system that guarantees privacy using anonymous credential schemes without a trusted third party. Similarly, Hohenberger *et al.* [HMPs14] advanced *ANONIZE*, a protocol for surveys that ensures authentication and privacy in presence of corrupted authorities. However, surveys have different security requirements than exams, for instance, surveys do not consider test authorship and fixed-term anonymity definitions.

Some related protocols have been proposed in the area of conference management systems. Kanav *et al.* [KLP14] introduced *CoCon*, a formally verified implementation of conference management system that guarantees confidentiality. Arapinis *et al.* [ABR12] introduced and formally analysed *ConfiChair*, a cryptographic protocol that addresses secrecy and privacy risks coming from a *malicious-but-cautious* cloud. Their work has been recently extended to support any cloud-based system that assumes honest managers, such as public tender management and recruitment process [ABR13]. In Remark!, a different attacker is considered since exam authorities, which are analogous to managers in cloud-based systems, can be corrupted.

5.2 Exponentiation mixnet

Remark! relies on ElGamal encryption, digital signature, and exponentiation mixnet. In this section, we detail the rudiments of exponentiation mixnet, while the reader can refer to chapter 3 for a brief description of ElGamal encryption and digital signature.

The main functionality provided by exponentiation mixnet is to generate a *pseudo* public key that allows the owner of the corresponding private key to encrypt and sign messages anonymously. An exponentiation mixnet takes in a batch of public keys and outputs a set of new pseudo public keys. The scheme ensures that no one but the owner of the public/private key pair can link a public key of the original batch with any of the pseudo public keys. In contrast to conventional re-encryption mixnet [Cha81] in which each term is independently re-encrypted, the peculiarity of exponentiation mixnet is that each mix server re-encrypts the terms by a common exponent value. This idea appeared first in the work of Haenni and Spycher [HS11].

In the following, we detail the construction of an exponentiation mixnet, which is depicted in Figure 5.1. Let g be a generator of a multiplicative subgroup $\mathbb{G}$ of order q. Let us assume n principals $\langle C_1, \dots C_n \rangle$, each have a pair of public/private keys (PK, SK) such that $PK = g^{SK}$. Let us assume m servers composing the exponentiation mixnet. The mix server mix_1 takes the batch of the public keys $\langle PK_1, \dots PK_n \rangle$, generates a fresh random $r_1 \in \{1, q-1\}$, and computes the batch of temporal pseudo public keys $\langle PK_1^{r_1}, \dots PK_n^{r_1} \rangle$. Then, mix_1 signs and sends to the bulletin board the computed batch in a secret shuffled order, namely the server posts $\langle PK^{\overline{r}_1}_{\overline{\pi}_1(1)}, \dots PK^{\overline{r}_1}_{\overline{\pi}_1(n)} \rangle$. Additionally, mix_1 posts a zero-knowledge proof of correctness and sends the new generator

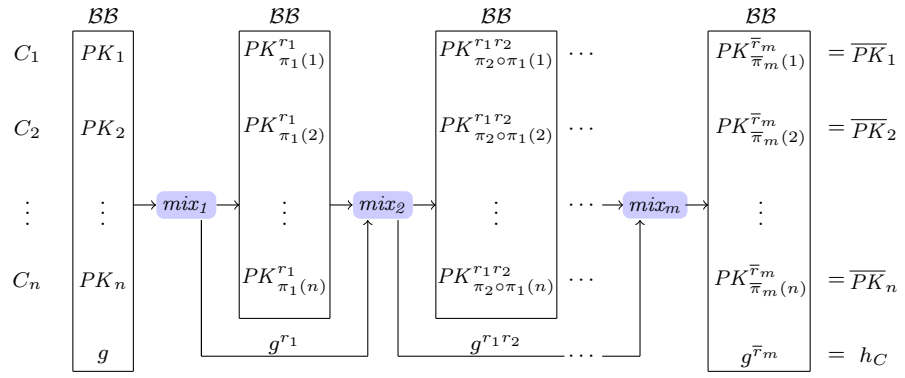

Figure 5.1: The exponentiation mixnet scheme

g^{r_1} to the next server over a secure channel. Further servers repeat the steps above as required. The last server, mix_m, publishes the final batch of pseudo public key $\langle PK^{\overline{r}_m}_{\overline{\pi}_m(1)}, \ldots PK^{\overline{r}_m}_{\overline{\pi}_m(n)}\rangle$ and the final generator $g^{\overline{r}_m}$, where $\overline{r}_m = \prod_{i=1}^{m} r_i$ and $\overline{\pi}_m = \pi_k \circ \cdots \circ \pi_1$. Note that the intermediate $g^{r_1}, \ldots, g^{\overline{r}_{m-1}}$ terms are not posted on the bulletin board. This prevents each principal to trace their intermediate pseudo public keys through the mixnet. Although it is not clear whether such eventuality is an attack, it is normally considered an undesired feature. Each principal C_i can find the corresponding pseudo public key using their private keys, since $g^{\overline{r}_m SK_i} = PK^{\overline{r}_m}_{\overline{\pi}_m(i)}$.

Remark! makes use of exponentiation mixnet at preparation to create the pseudonyms for candidates and examiners. The mixnet is also required at notification to revoke the pseudonyms of candidates. In so doing, each server mix_i reveals its random value r_i, hence by revealing all the values $\overline{r}_m$ it is possible to link the pseudonyms to the identities of the candidates.

5.3 Description

Remark! has four roles: exam authority (EA), candidate (C), examiner (E), and mixnet (NET). The exam authority manages the exam and also plays the roles of collector and notifier.

Remark! relies on a bulletin board to publish pseudonyms, questions, tests, and marks. As we discussed in the previous section, the bulletin board is also used in the exponentiation mixnet scheme. In the remainder, we assume that anyone can post messages on the bulletin board, even the attacker. Thus, we require each principal to sign their messages. However, if one assumes that the bulletin board has appropriate write access control mechanisms, namely it only publishes messages that originate from eligible principals, signatures may not be necessary.

In the following, we detail Remark! according the phases of an exam. Figure 5.2 illustrates the protocol's steps in form of a message sequence chart.

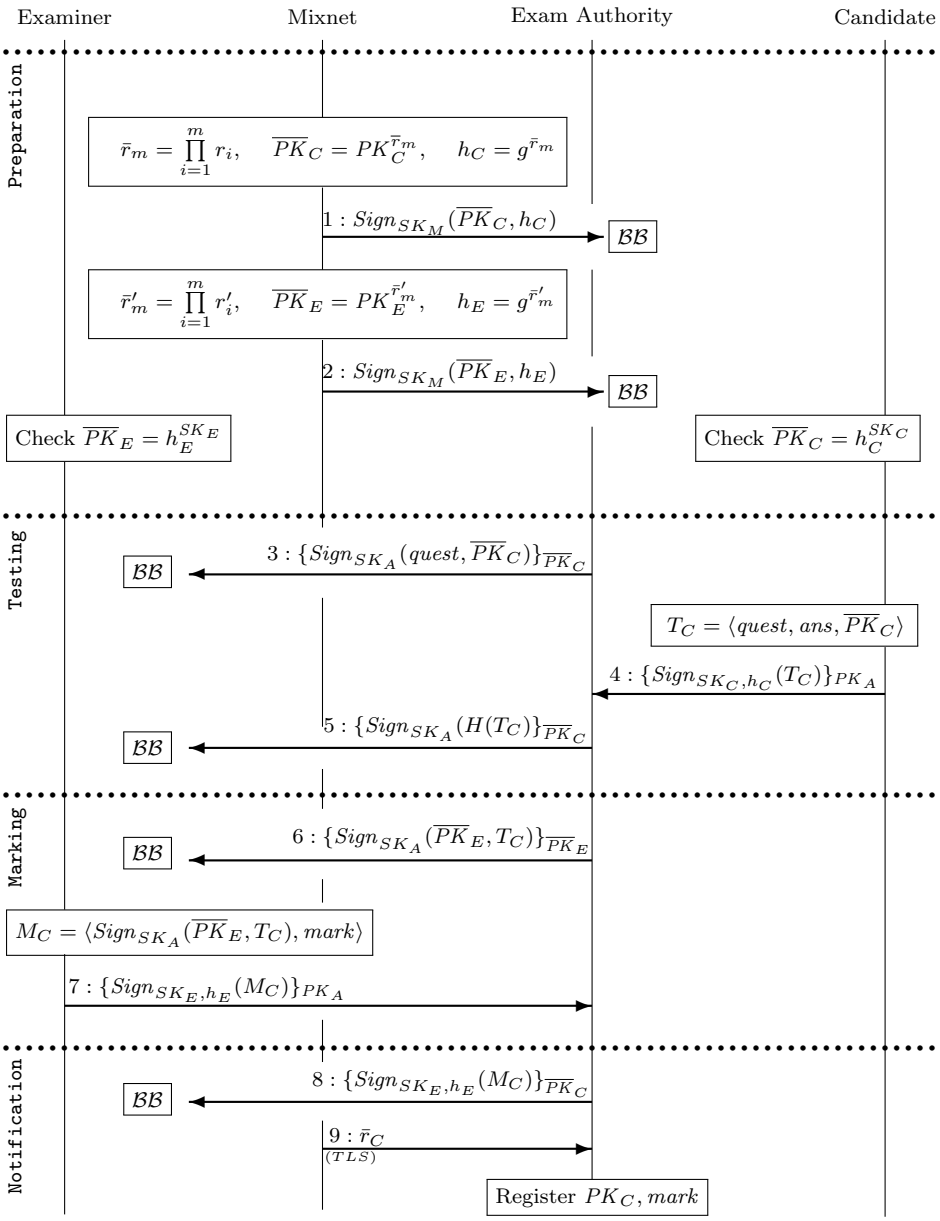

Figure 5.2: The Remark! Internet-based exam protocol

Preparation

The exponentiation mixnet generates the pseudonyms of candidate and examiner. The generation takes place in two independent runs: first the mixnet generates the pseudonym of candidates and then the pseudonym of examiners. Such separation is necessary because only the identities of candidates should be revealed at notification.

The public key PK_C of an eligible candidate C is processed by the exponentiation mixnet among the public keys of other candidates. After the last mix server publishes the list of pseudonyms and the new generator $h_C := g^{\overline{r}_m}$, the candidate identifies her pseudo public key $\overline{PK}_C$ computing $h_C^{SK_C}$. The pseudo public key from now on serves as the pseudonym for C.

After the pseudonyms of candidates have been published (step 1), the mixnet generates the pseudonyms for examiners in a similar way. Since the mixnet generates the pseudonyms of examiners using different random values, a new generator h_E is published at the end of the mix (step 2).

Testing

The exam authority generates the questions, signs them with its private key SK_A, and encrypts each question under a candidate pseudonym. We do not specify how the exam authority generates the questions in order to include different forms of questions (e.g., multiple choice, free-response, etc.) and assignments (e.g., single question, different questions for candidate, random permutations of a set of questions, etc.). In the remainder, with *question* we actually refer to a list of questions possibly of size one.

Remark! assumes that an invigilator authenticates the candidate by checking whether the personal details printed on the top of the smart card matches the candidate identity. For remote authentication, this procedure can be supported with tools such as ProctorU. Then, the exam authority publishes the encrypted questions on the bulletin board (step 3). After the candidate answers the test, she appends the answer to her pseudonym and question, so the filled test consists of $T_C = \langle ques, ans, \overline{PK}_C \rangle$. Then, she signs the test T_C with her private key SK_C using the generator h_C. Thus, the signature can be verified using the pseudonym of the candidate $\overline{PK}_C$ with respect to h_C. The candidate then encrypts the signed test with the public key of the exam authority PK_A, and submits it (step 4). The exam authority decrypts the test, and then signs the hash of T_C using its private key SK_A. It then encrypts the signed hash under the corresponding candidate's pseudonym, that is, $\{Sign_{SK_A}(H(T_C))\}_{\overline{PK}_C}$, and publishes such encryption as receipt (step 5).

Marking

The exam authority randomly chooses an eligible examiner pseudonym $\overline{PK}_E$, and encrypts the signed test T_C under the chosen examiner pseudonym (step 6). Note that the exam authority does not know the real identity of the examiner. Moreover, it is possible to introduce a universally verifiable deterministic assignment of test to examiners. For example, encrypted tests and the examiner pseudonyms could be posted in two lexically ordered lists, and the exam authority cyclically assigns a test to an examiner according the order.

After the designated examiner marks the test, he appends the mark to the signed test, thus generating the evaluation $M_C = \langle Sign_{SK_A}(\overline{PK}_E, T_C), mark \rangle$. The examiner then signs M_C with his private key SK_E and the generator h_E, and encrypts the signed evaluation under the public key of the exam authority PK_A (step 7).

Notification

The exam authority receives from the examiner the encrypted evaluation, which it decrypts and re-encrypts under the corresponding candidate pseudonym $\overline{PK}_C$. After the exam authority publishes all the test evaluations (step 8), it asks the mixnet to reveal the random values r used to generate the pseudonyms of the candidates (step 9). In so doing, the candidate anonymity is revoked, and the mark can finally be registered. Note that each candidate learns the corresponding mark before $\overline{r}_m$ is revealed.

5.4 Formal Analysis of Authentication and Privacy

We analyse Remark! in ProVerif. We consider the authentication and privacy requirements formally specified in chapter 3. The requirements with short descriptions are recalled below:

- Authentication
  - *Candidate Authorisation*, which says that only registered candidates can take the exam.
  - *Answer Authenticity*, which says that the collector considers only the answers that candidates actually submitted.
  - *Test Origin Authentication*, which says that the collector accepts only tests that originate from registered candidates.
  - *Test Authenticity*, which says that the examiner only marks the tests intended for him.
  - *Mark Authenticity*, which says that the candidate receives the mark assigned to her test by the examiner chosen by the collector.
- Privacy
  - *Question Indistinguishability*, which says that the questions are not revealed until testing begins.
  - *Anonymous Marking*, which says that the examiner marks a test while ignoring its author.
  - *Anonymous Examiner*, which says that the candidate cannot learn which examiner marked her test.
  - *Mark Privacy*, which says that no one learns the marks, besides the examiner, the concerned candidate, and the notifier.
  - *Mark Anonymity*, which says that no one learns the association between a mark and the corresponding candidate.

Primitive	Equation
ElGamal encryption	$decrypt(encrypt(m, pk(k), r), k) = m$ $decrypt(encrypt(m, pseudo\_pub(pk(k), rce), r), pseudo\_priv(k, exp(rce))) = m$ $checkpseudo(pseudo\_pub(pk(k), rce), pseudo\_priv(k, exp(rce))) = true$
Digital signature	$getmess(sign(m, k)) = m$ $checksign(sign(m, k), spk(k)) = m$ $checksign(sign(m, pseudo\_priv(k, exp(rce))), pseudo\_pub(pk(k), rce)) = m$

Table 5.1: Equational theory to model Remark!

Model choices

We model the bulletin board as a public channel, and use the equational theory depicted in Table 5.1. The theory consists of the standard equations for ElGamal encryption and digital signatures extended with novel equations that model pseudonyms as public keys. The pseudonym, which also serves as test identifier, can be generated using the function *pseudo_pub*, which takes in a public key and a random exponent. In fact, this function models the main feature of exponentiation mixnet. The function *pseudo_priv* can be used by a principal to decrypt or sign anonymous messages. The function takes in the private key of the principal and the new generator published by the mixnet. The function *checkpseudo* allows a principal to check whether a pseudonym is associated with the principal's private key. In practice, principals use this function to identify their pseudonyms published on the bulletin board.

The process of the exam authority is in Figure 5.3, the process of the mixnet is in Figure 5.5, the process of the candidate is in Figure 5.4, and the process of examiner is in Figure 5.6. The exam process is depicted in Figure 5.7. In each process we replace the identity of candidate with the corresponding candidate's pseudonym inside the events. This choice is sound because the equational theory preserves the bijective mapping between keys and pseudonyms.

We analyse Remark! in ProVerif with the same approach used to verify the Huszti-Pethő protocol in chapter 3. In particular, we use ProVerif's `noninterf` and `choice[]` commands to verify the privacy requirements. The full ProVerif code is available on the Internet [Giu15].

Results

Assuming an attacker in control of the network and honest principals, ProVerif successfully proves all authentication and privacy requirements. Table 5.2 reports the execution times over an Intel Core i7 3.0 GHz machine with 8 GB RAM. Also assuming corrupted principals, ProVerif proves that Remark! ensures all the requirements. Table 5.2 also reports the honest roles that are required for each requirement to hold. Note that we only model the processes needed to specify the requirement. For example, the specification of Anonymous

```
let EA (skA:skey, pkN:pkey, ques:bitstring) =
(*Preparation*)
 in(bbn, (pseudo_C:pkey, hc:bitstring, r: role, spseC:bitstring));
 if (pseudo_C, hc, r) = checksign(spseC, pkN) && r = C then
  let sques = sign(ques, skA) in
  let eques = encrypt( (ques,sques), pseudo_C) in
  out(bba, eques);
  in(ch, eca:bitstring);
  let ((=ques, ans:bitstring, =pseudo_C), sca:bitstring) =
  decrypt(eca, skA) in
  if (ques, ans, pseudo_C) = checksign(sca, pseudo_C) then
   (* EA collects the test from C with pseudonym pseudo_C *)
   event collected(pseudo_C, ques, ans);
   let ca = (ques, ans, pseudo_C) in
   let sca' = sign(ca, skA)  in
   let eca' = encrypt((ca, sca'), pseudo_C) in
   out(bba, eca');

(* Marking *)
   in(bbn, (pseudo_E:pkey, he:bitstring, rolet:role,
      spseE:bitstring));
   if (pseudo_E, he, rolet) = checksign(spseE, pkN) &&
      rolet = E then
    let ca'' = (ques, ans, pseudo_C, pseudo_E) in
    let sca'' = sign(ca'', skA)  in
    let eca'' = encrypt((ca'', sca''), pseudo_E) in
    (* EA distributed the test (pseudo_C,ques,ans) *)
    (* identified by pseudo_C (id_form = pseudo_C) to E pseudo_E*)
    event distributed(pseudo_C,ques,ans,pseudo_C,pseudo_E);
    out(bba, eca'');
    in(ch, ema:bitstring);
    let ((=ca'', =sca'', mark:bitstring), sma:bitstring) =
    decrypt(ema, skA) in
    if (ca'', sca'', mark) = checksign(sma, pseudo_E)  then

(* Notification *)
     let ma = (ca'', sca'', mark) in
     let ema' = encrypt((ma, sma), pseudo_C) in
     out(bba, ema');
     (*Reveal ID*)
     in(ch, encsignetX: bitstring).
```

Figure 5.3: The process of the exam authority

```
let Cand (skC:skey, pkA:pkey, pkN:pkey, ans:bitstring) =
(*Preparation *)
 in(bbn, (pseudo_C:pkey, hc:bitstring, r: role, spseC:bitstring));
 if (pseudo_C, hc, r) = checksign(spseC, pkN) && r = C then
  let priv_C = pseudo_priv(skC, hc) in
  if checkpseudo(pseudo_C, priv_C) =true then

(*Testing*)
  in(bba, eques:bitstring);
  let (ques: bitstring, sques:bitstring)=decrypt(eques, priv_C) in
  if ques=checksign(sques, pkA) then
   let ca = (ques, ans, pseudo_C) in
   let sca = sign(ca,priv_C) in
   let eca = encrypt((ca, sca), pkA) in
   (* C with pseudo_C  submits his test (ques, ans) *)
   event submitted(pseudo_C, ques, ans);
   out(ch, eca);
   in(bba, eca':bitstring);
   let (=ca, sca':bitstring) = decrypt(eca', priv_C) in
   if (ques, ans, pseudo_C) = checksign(sca', pkA) then

(*Notification*)
    in(bba, ema':bitstring);
    in(bbn, (pseudo_E:pkey, he:bitstring, role_E: role,
       spseE:bitstring));
    if (pseudo_E, he, E) = checksign(spseE, pkN) then
     let ((ca'': bitstring, sca'': bitstring, mark:bitstring),
           sma:bitstring) = decrypt(ema', priv_C) in
     if ca''=(ques, ans, pseudo_C, pseudo_E) &&
        ca''=checksign(sca'', pkA) then
      if ((ques, ans, pseudo_C, pseudo_E), sca'', mark) =
         checksign(sma, pseudo_E) then
       (* C with pseudo_C is notified with "mark" *)
       event notified(pseudo_C, mark).
```

Figure 5.4: The process of the candidate

```
let NET (skN:skey, pkA:pkey, rc:bitstring) =
 in(ch, (R: role));
 get publickey(=R, rx, pkX) in
 let hx = exp(rx) in
 let pseudo_X = pseudo_pub(pkX,rx) in
 let spseX = sign ((pseudo_X, hx, R), skN) in
 out(bbn, ( pseudo_X, hx, R, spseX));
 (*Reveal rc*)
 let signetX = sign( (rc), skN) in
 let encsignetX = encrypt ( (rc, signetX), pkA) in
 out(ch, encsignetX).
```

Figure 5.5: The process of the mixnet

```
let Ex (skE:skey, pkA:pkey, pkN:pkey, mark:bitstring) =
(*Preparation *)
 in(bbn, (pseudo_E:pkey, he:bitstring, r: role, spseE:bitstring));
 if (pseudo_E, he, E) = checksign(spseE, pkN) then
  let priv_E = pseudo_priv(skE, he) in
  if checkpseudo(pseudo_E, priv_E) =true then

(* Marking *)
   in(bba, eca'':bitstring);
   let ((ques:bitstring, ans:bitstring, pseudo_C:pkey, =pseudo_E),
       sca':bitstring) = decrypt(eca'', priv_E) in
   if (ques, ans, pseudo_C,pseudo_E) = checksign(sca', pkA) then
    let ca = (ques, ans, pseudo_C, pseudo_E) in
    let ma:bitstring = (ca, sca', mark) in
    let sma:bitstring =  sign(ma,priv_E) in
    let ema = encrypt((ma, sma), pkA) in
    event marked(ques,ans,mark,pseudo_C,pseudo_E);
    (* E with  pseudo_E marked the test (ques, ans) *)
    (* identified by pseudo_C with mark *)
    out(ch, ema).
```

Figure 5.6: The process of the examiner

```
process
 !(
 (*Products of the secret exponent values of the servers *)
 (* (represented by the NET): rc for C and re for E *)
 new rc: bitstring;
 new re: bitstring;
 (*Assume one NET and one EA*)
 new skA: skey; let pkA = pk(skA) in out (ch, pkA);
 new skN: skey; let pkN = pk(skN) in out (ch, pkN);

 (!( NET(skN, pkA, rc))) |
 (!( new ques:bitstring;  EA(skA, pkN, ques))) |
 (!( new skC: skey; let pkC = pk(skC) in out (ch, pkC);
     new ans:bitstring; insert publickey(C, rc, pkC);
     Cand(skC, pkA, pkN, ans))
   ) |
 (!( new skE: skey; let pkE = pk(skE) in out (ch, pkE);
     new mark:bitstring; insert publickey(E, re, pkE);
     Ex(skE, pkA, pkN, mark))
   )

 )
```

Figure 5.7: The exam process

Requirement	Result	Time	Honest roles
Candidate Authorisation	✓	1 s	(C, EA, NET)
Answer Authenticity	✓	1 s	(E, EA, NET)
Test Origin Authentication	✓	1 s	(NET)
Test Authenticity	✓	1 s	(E, EA, NET)
Mark Authenticity	✓	1 s	(E, EA, NET)
Question Indistinguishability	✓	1 s	(E, EA, NET)
Anonymous Marking	✓	1 s	(C, NET)
Anonymous Examiner	✓	1 s	(E, NET)
Mark Privacy	✓	3 m 39 s	(EA, NET)

Table 5.2: Summary of authentication and privacy analysis of Remark!

Marking requires two candidates to be honest, otherwise they could just reveal their tests to the attacker, who would trivially violate the protocol. However, all other candidates can be corrupted and collude with the attacker to violate the protocol.

Notably, Remark! ensures a stronger version of Anonymous Examiner since no one, even the exam authority, knows which examiner marks which test. It can be observed that Mark Anonymity is not in Figure 5.2: since Remark! ensures Mark Privacy, it also guarantees Mark Anonymity.

Remark. We report an issue on an early version of Remark! that witnesses how formal approaches contribute to achieve a deep understanding of the design models. In the first draft of Remark!, the receipt of submission of a test T_C consisted of the message $\{Sign_{SK_A}(T_C)\}_{\overline{PK}_C}$, that is, the exam authority signs the test and posts the signed test encrypted with the candidate's pseudonym. Moreover, the assignment of the test to the examiner consisted of the message $\{Sign_{SK_A}(T_C)\}_{\overline{PK}_E}$, namely the signed test encrypted with an eligible examiner's pseudonym. The rest of the protocol was unchanged respect to the current version. With these two modifications ProVerif cannot prove Test Authenticity. In fact, the attack trace shows that a corrupted candidate can pick an examiner of her choice by re-encrypting the signed receipt received from the exam authority. It means that the candidate can influence the choice of the examiner who marks her test. Such attack could be avoided assuming an access control mechanism that would not allow the candidate to post on the bulletin board.

However, the fixes implemented in the final version of Remark! shows that there is no need of access control mechanisms to secure the protocol. The first fix consists in signing the hash of the test as receipt. The second fix consists in making the pseudonym of the chosen examiner explicit. In doing so, the signature of the exam authority within the receipt cannot be used by a candidate to designate any examiner.

5.5 Formal Analysis of Verifiability

The ProVerif model proposed to check authentication and privacy in the previous section can be also used to analyse Remark! for verifiability. The definitions

of authentication and privacy introduced in chapter 3 are formulated in the applied π-calculus and can be coded straight to ProVerif. Conversely, verifiability definitions are expressed in a more abstract model. Thus, it is necessary to map sets and relations specified in the verifiability model to Remark!. We briefly recall the informal descriptions of the verifiability requirements specified in chapter 4.

- Individual Verifiability: a candidate can check that
  - *Question Validity*: she received the questions actually generated by the question committee.
  - *Marking Correctness*: the mark she received is correctly computed on her test.
  - *Test Integrity*: her test is accepted and marked as she submitted it.
  - *Test Markedness*: the test she submitted is marked without modification.
  - *Marking Integrity*: the mark attributed to her test is assigned to her without any modification.
  - *Marking Notification*: she received the mark assigned to her.
- Universal Verifiability: an auditor can check that
  - *Registration*: all accepted tests are submitted by registered candidates.
  - *Marking Correctness*: all the marks attributed by the examiners to the tests are computed correctly.
  - *Test Integrity*: all and only accepted tests are marked without any modification.
  - *Test Markedness*: only the accepted tests are marked without modification.
  - *Marking Integrity*: all and only the marks associated to the tests are assigned to the corresponding candidates with no modifications.

We recall that Definition 28, which we introduced in chapter 4, considers the data sets I, Q, A, M, and their elements i, q, a, m, which specify the candidate identities, the questions, the answers, and the marks respectively. In Remark!, the set I contains the candidate pseudonyms rather than the identities. In the previous section we argued this choice to be sound. The sets Q, A, and M contains the messages that correspond to questions, answers and marks generated by the protocol's principals, possibly manipulated by the attacker.

The relations `Accepted`, `Marked`, and `Assigned` are built from the posts that appear on the bulletin board. The tuples $(i, (q, a))$ of the relation `Accepted` consist of the receipts of submission that the exam authority publishes on the bulletin board at the end of testing. The tuples $(i, (q, a), m)$ of the relation `Marked` coincide with the tuples (i, m) of `Assigned`, and consist of the messages that the exam authority publishes on the bulletin board at marking. Precisely, the tuples $(i, (q, a), m)$ are generated from the marked test signed by the examiner, that is, $Sign_{SK_E, h_E}(M_C)$. The tuples (i, m) instead are built from the encryption of the marked test generated by the exam authority, that is, $\{Sign_{SK_E, h_E}(M_C)\}_{\overline{PK}_C}$.

It can be observed that encryption under the candidate's pseudonym officially assigns the mark to the candidate.

Finally, the function `Correct`, which is the algorithm used to mark the tests, can be modelled as a ProVerif table.

5.5.1 Individual Verifiability

Individual verifiability definitions require the existence of verifiability-tests that candidates run to check the properties of the protocol. We show that Remark! has the necessary verifiability-tests. In ProVerif, we model the verifiability-tests as processes that emit the event `OK` when the verifiability-test succeeds, and emit the event `KO` when the verifiability-test fails.

We use correspondence assertions to prove soundness. ProVerif checks verifiability as a reachbility property. The verification strategy normally consists of checking that the event `OK` is always preceded by the event emitted in the part of the code where the predicate becomes satisfied. In the ProVerif model of Remark! we assume an honest candidate principal who plays the role of the verifier. The other principals are usually corrupted, if not stated otherwise. The verifiability-test receives the data from the candidate via a private channel, and the remaining data posted on the bulletin board via public channels. This allows an attacker to manipulate the input data. Corrupted principals may collude with the attacker.

We resort to unreachability of the event `KO` to prove completeness. In this case, the ProVerif model enforces only honest principals and prevents the attacker to manipulate the input data of the verifiability-tests. In fact, a complete verifiability-test must succeed if its input data is correct.

In the following paragraphs, we specify the verifiability-tests for Remark!. We also discuss the conditions to achieve sound and complete verifiability-tests according to each individual verifiability requirement.

Question Validity

Remark! assumes that the exam authority generates the questions at preparation and publishes them at testing. Thus, we model the exam authority as an honest process in ProVerif, otherwise a corrupted exam authority would publish questions that are different from the ones actually generated.

The verifiability-test *testQV* , which is depicted in Figure 5.8, receives the question *eques* that is published on the bulletin board from a public channel. It also receives the candidate's question *ques* and private key *priv_C* from a private channel. The verifiability-test checks whether the candidate actually received the question published by the exam authority on the bulletin board.

To prove soundness, we annotate the ProVerif process of the exam authority with the event `generated` where the questions are generated. Then, ProVerif checks if the verifiability-test emits the event `OK` only if the exam authority actually generated the question received from the candidate, namely ProVerif checks the following correspondence assertion:

$$\texttt{OK}\langle ques \rangle \rightsquigarrow \texttt{generated}\langle ques \rangle$$

To prove completeness, ProVerif checks that the verifiability-test process does not emit the event `KO` when the input data is correct. ProVerif confirms

```
let testQV(pkA: pkey, pch: channel) =
 in(bba, eques:bitstring);
 in(pch, (ques: bitstring, priv_C: skey));

 let (ques':bitstring, sques:bitstring) =
     decrypt(eques, priv_C) in
 let (ques'':bitstring,p seudoC:bitstring) =
     checksign(sques, pkA) in

 if ques'=ques && ques''=ques' then event OK
 else event KO.
```

Figure 5.8: The Question Validity individual verifiability-test

the verifiability-test is sound and complete, so we can conclude that Remark! is question validity verifiable.

Marking Correctness

Remark! is designed to support different forms of questions (e.g., multiple choice, free-response, etc.), hence there is no universal marking algorithm that can be used to evaluate the answers. However, we can assume that the exam authority publishes the table of evaluations that maps an answer to a mark after the exam concludes. We thus model the exam authority as an honest process in ProVerif to check Marking Correctness.

The verifiability-test *testMC*, which is in Figure 5.9, receives the test (*ques*, *ans*) submitted by the candidate, and the mark *mark* notified to her. The verifiability-test checks if the mark reported on the table of evaluations and associated to the candidate's answer coincides with the mark received from the candidate.

To prove soundness, we annotate the ProVerif process of the candidate with the event `correct` where the candidate receives the mark at notification. ProVerif checks that if the verifiability-test emits the event `OK`, then a previous event `correct` was emitted. This is formalised as:

$$\texttt{OK}\langle ques, ans, mark\rangle \rightsquigarrow \texttt{correct}\langle ques, ans, mark\rangle$$

ProVerif checks that the verifiability-test does not emit the event `KO` to prove completeness.

Thus, assuming an honest exam authority that provides the table of evaluations at end of exam, Remark! is marking correctness verifiable.

Test Integrity

The verifiability-test *testTI* in Figure 5.10 takes in the test (*ques*,*ans*) submitted by the candidate via a private channel, and the receipt of submission *eca'* and the notification *ema'* published on the bulletin board via a public channel. The verifiability-test checks if candidate's submission, the receipt, and the notification contain the same question, answer, and pseudonym.

```
let testMC (pkA: pkey, priv_ch: channel) =
 in(priv_ch, (ques: bitstring, ans: bitstring, mark: bitstring));

 get correct_ans(=ques,=ans,mark':bitstring) in

 if mark'=mark then
  event OK
 else KO.
```

Figure 5.9: The Marking Correctness individual verifiability-test

```
let testTI (pkA: pkey, priv_ch: channel) =
 in(priv_ch, (priv_C: skey, ques: bitstring, ans: bitstring,
              pseudo_C: pkey));
 in(bbn, (pseudo_E:pkey, he:bitstring, rolet: role,
          spseE:bitstring));
 in(bba, eca': bitstring);
 in(bba, ema':bitstring);

 (* If the message on the BB is signed by the authority, *)
 (* it is considered as part of the relation Accepted. *)
 let (ca: bitstring, sca':bitstring) = decrypt(eca', priv_C) in
 let (ques': bitstring, ans': bitstring, pseudo_C': pkey) =
     checksign(sca', pkA) in
 (* If the message on the BB is signed by the examiner, *)
 (* it is considered as part of the relation Marked. *)
 let (((ques'': bitstring, ans'': bitstring, pseudo_C'': pkey),
       sca1: bitstring, mark:bitstring), sma:bitstring) =
     decrypt(ema', priv_C) in
 let ((ques''': bitstring, ans''': bitstring, pseudo_C''': pkey),
      sca1': bitstring, mark':bitstring) =
     checksign(sma, pseudo_E) in

 if ques'=ques && ans'=ans && pseudo_C'=pseudo_C && ques''=ques &&
    ans''=ans && pseudo_C''=pseudo_C && (ques', ans', pseudo_C') =
    checksign(sca1,pkA) && ques'''=ques && ans'''=ans &&
    pseudo_C'''=pseudo_C && sca1'=sca1 then
  event OK
 else event KO.
```

Figure 5.10: The Test Integrity individual verifiability-test

To prove soundness, we annotate the verifiability-test with the events `accepted` and `marked` that map the corresponding relations. In particular, the receipt of submission is part of the relation `Accepted` if it is signed by the exam authority and encrypted under the pseudonym of the candidate. Similarly, the notification is part of the relation `Marked` if it is signed by the examiner and encrypted under the pseudonym of the candidate. The requirement can be formalised with the following correspondence assertion:

$$\texttt{OK}\langle id, ques, ans\rangle \rightsquigarrow \texttt{marked}\langle id, ques, ans\rangle \cup \texttt{accepted}\langle id, ques, ans\rangle$$

To prove completeness, ProVerif checks that the verifiability-test process does not emit the event KO when the input data is correct.

ProVerif shows that the verifiability-test for Test Integrity is sound and complete. Note that a corrupted exam authority can publish two different receipts for the same test on the bulletin board. However, since the bulletin board is append-only, the candidate notices if the exam authority appends two different receipts for her submission because only the candidate knows the private key.

Test Markedness

Since Remark! has a sound and complete verifiability-test for Test Integrity, we can build from this a sound and complete verifiability-test for Test Markedness. It is sufficient to not consider the receipt of submission as input, and just check whether the candidate's submission and the data obtained from notification contain the same question, answer, and pseudonym. The verifiability-test *testTM* is depicted in Figure 5.11. To prove soundness, it is sufficient to prove the following correspondence assertion:

$$\texttt{OK}\langle id, ques, ans\rangle \rightsquigarrow \texttt{marked}\langle id, ques, ans\rangle$$

Mark Integrity

The verifiability-test *testMI* in Figure 5.12 takes in the test (*ques*,*ans*) submitted by the candidate via a private channel, and the notification *ema'* published by the exam authority on the bulletin board. The verifiability-test checks if the test provided by the candidate and the notification on the bulletin board contain the same question, answer, and pseudonym, and if the examiner's signature on the mark is correct.

To check soundness in ProVerif, we annotate the verifiability-test with the events `assigned` and `marked` that map the corresponding relations. The data on the notification message is part of the relation `Assigned` if the data is signed by the exam authority and encrypted under the pseudonym of the candidate. This data is also part of the relation `Marked` if it also include the signature of the examiner. The correspondence assertion to check soundness is:

$$\begin{aligned}\texttt{OK}\langle id, ques, ans, mark\rangle \rightsquigarrow\ & \texttt{marked}\langle id, ques, ans, mark\rangle \cup \\ & \texttt{assigned}\langle id, ques, ans, mark\rangle\end{aligned}$$

We check completeness as usual, and ProVerif confirms that the verifiability-test for Mark Integrity is sound and complete.

```
let testTM (pkA: pkey, priv_ch: channel) =
 in(priv_ch, (priv_C: skey,ques: bitstring, ans: bitstring,
              pseudo_C: pkey));
 in(bbn, (pseudo_E:pkey, he:bitstring, rolet: role,
          spseE:bitstring));
 in(bba, ema:bitstring);

 (* If the message on the BB is signed by the examiner, *)
 (* it is considered as part of the relation Marked. *)
 let (((ques': bitstring, ans': bitstring, pseudo_C': pkey),
       sca1: bitstring, mark:bitstring), sma:bitstring) =
     decrypt(ema, priv_C) in
 let ((ques'': bitstring, ans'': bitstring, pseudo_C'': pkey),
      sca1': bitstring, mark':bitstring) =
     checksign(sma, pseudo_E) in

 if ques'=ques && ans'=ans && pseudo_C'=pseudo_C &&ques''= ques &&
    ans''=ans && pseudo_C''=pseudo_C && (ques', ans', pseudo_C') =
    checksign(sca1,pkA) && sca1'=sca1 then
  event OK
 else event KO.
```

Figure 5.11: The Test Markedness individual verifiability-test

```
let testMI (pkA: pkey, priv_ch: channel) =
 in(priv_ch, (priv_C: skey,ques: bitstring, ans: bitstring,
    pseudo_C: pkey));
 in(bbn, (pseudo_E:pkey, he:bitstring, rolet: role,
    spseE:bitstring));
 in(bba, ema':bitstring);

 (* Assigned is the mark sent by the authority *)
 let (((ques': bitstring, ans': bitstring, pseudo_C': pkey),
       sca': bitstring, mark:bitstring), sma:bitstring) =
     decrypt(ema', priv_C) in
 (* Marked are the marks signed by the examiner *)
 let ((ques'': bitstring, ans'': bitstring, pseudo_C'': pkey),
      sca'': bitstring, mark': bitstring) =
     checksign(sma, pseudo_E) in

 if ques'=ques && ans'=ans && pseudo_C'=pseudo_C && ques''=ques &&
    ans''=ans && pseudo_C''=pseudo_C && mark=mark' &&
    (ques', ans', pseudo_C')=checksign(sca',pkA) then
  event OK
 else KO.
```

Figure 5.12: The Mark Integrity individual verifiability-test

```
let testMNI (pkA: pkey, priv_ch: channel) =
 in(priv_ch, (priv_C: skey,mark: bitstring, pseudo_C: pkey,
              ema': bitstring));
 in(bba, ema: bitstring);

 (* Assigned is the mark sent by the authority *)
 let (((ques:bitstring, ans: bitstring, pseudo_C':pkey),
       sca: bitstring, mark':bitstring), sma:bitstring) =
     decrypt(ema, priv_C) in

 if (ques,ans,pseudo_C')=checksign(sca, pkA) &&
    pseudo_C'=pseudo_C &&  mark'=mark then
  event OK
 else event KO.
```

Figure 5.13: The Mark Notification Integrity individual verifiability-test

Requirement	Soundness	Completeness
Question Validity	✓ (EA)	✓(all)
Test Integrity	✓	✓(all)
Test Markedness	✓	✓(all)
Marking Correctness	✓ (EA)	✓(all)
Mark Integrity	✓	✓(all)
Mark Notification Integrity	✓	✓(all)

Table 5.3: Summary of the analysis of Remark! for I.V. requirements

Mark Notification Integrity

The last individual verifiability definition concerns the check of the integrity of the notified mark. The verifiability-test *testMNI* in Figure 5.13 is fed via a private channel with the mark *mark* that the candidate received at notification. The verifiability-test *testMNI* also takes in the official notification *ema'* published in the bulletin board via a public channel, and checks if the mark provided in the notification coincides with the one received from the candidate.

Similarly to Mark Integrity, we annotate the verifiability-test *testMNI* with the event `assigned` to prove soundness. ProVerif checks if the verifiability-test emits the event `OK` only if the mark notified to the candidate is the same officially assigned at the end of the exam. This is formalised as:

$$\texttt{OK}\langle id, mark\rangle \rightsquigarrow \texttt{assigned}\langle id, mark\rangle$$

Also in this case ProVerif checks that the verifiability-test process does not emit the event KO when the input data is correct to prove completeness. ProVerif confirms the verifiability-test is sound and complete, hence Remark! is mark notification integrity verifiable.

Table 5.3 summarises the results of the individual verifiability analysis of Remark! and reports the roles required to be honest.

5.5.2 Universal verifiability

Also for the specification of universal verifiability definitions, we model verifiability-tests as processes that emit the event `OK` when the verifiability-test succeeds, and emit the event `KO` when the verifiability-test fails.

In the case of universal verifiability an auditor runs the verifiability-tests, namely the auditor plays the role of the verifier. This requires a different approach to prove soundness compared to the approach used for individual verifiability definitions, in which the candidate plays as verifier. In fact, in the case of universal verifiability also the candidate can be corrupted, hence it can be hard to find a ProVerif process that can be annotated with events to check soundness via correspondence assertions.

The different approach consists of proving soundness of the verifiability-tests using unreachability of the event `KO`. The underlying idea is that every time the verifiability-test succeeds, which means that it emits the event `OK`, we check if the decryption of the concerned ciphertext gives the expected plaintext. If not, the event `KO` is emitted, thus the verifiability-test is not sound.

As we shall see later, it is however possible to prove the soundness of the verifiability-test for Registration requirement using correspondence assertions. This is possible because the NET is assumed to be honest, hence the corresponding ProVerif process can be annotated with an event that is emitted when registration concludes. We always use unreachability of the event `KO` to prove completeness of the verifiability-tests.

Remark. It can be noted that all messages posted by the exam authority on the bulletin board are encrypted under the pseudonym of either the candidate or the examiner, hence no public data can be used as it is by the auditor. Candidates and examiners hold long-term pairs of public/private keys, and it is implausible that they reveal their private keys for audit purposes. Since the auditor cannot decrypt a ciphertext message posted on the bulletin board, the auditor should be rather provided with the corresponding plaintext and pseudonym. In so doing, the auditor can encrypt the plaintext with the pseudonym and check if the encryption coincides with the same ciphertext message posted on the bulletin board. Since Remark! uses ElGamal encryption, which is probabilistic, the auditor should be also provided with the random value used to encrypt a message.

In the following, we specify the data that the exam authority should provide to the auditor after the exam concludes.

- Registration: the exam authority reveals the signatures inside the receipts $receipt = \{Sign_{SK_A}(H(T_C)\}_{\overline{PK_C}}$ posted on the bulletin board and the random values used to encrypt the receipts.

- Marking Correctness: the exam authority reveals the marked tests inside the evaluations $sma = \{Sign_{SK_E,h_E}(M_C)\}_{PK_A}$, the random values used to encrypt the marked tests, and the table *correct_ans* that maps each mark to each answer.

- Test Integrity: the exam authority reveals the marked tests inside the evaluations, the random values used to encrypt the marked tests, plus the data disclosed for Registration.

- Test Markedness: the exam authority reveals the same data disclosed for Test Integrity.
- Mark Integrity: the exam authority reveals the examiners' signatures on the marked tests inside the evaluations, and the random values used to encrypt the notifications $notif = \{Sign_{SK_E,h_E}(M_C)\}_{\overline{PK}_C}$ before posting them on the bulletin board.

We anticipate that it is not possible to automatically prove the universal verifiability requirements in ProVerif. To prove such requirements it is needed to iterate over all candidates, but ProVerif does not support loops. We thus prove the base case of each requirement automatically in ProVerif, in which it is considered only one accepted test or one assigned mark. Then, we provide manual induction proofs that generalise the ProVerif result to the general case with an arbitrary number of candidates.

Registration

The verifiability-test *testUR*, which is depicted in Figure 5.14, takes in from the bulletin board the pseudonyms of the candidates signed by the mixnet and the receipts of submissions generated by the exam authority. In so doing, the auditor can check that the exam authority accepted only tests signed with pseudonyms posted by the mixnet during preparation.

ProVerif proves that the verifiability-test is complete and sound for the base case, which considers one accepted test and an unbounded number of candidates. To prove the general case, namely for an unbounded number of accepted tests and candidates, it is necessary to show that

$$testUR(E) = true \Leftrightarrow \{i : (i, x) \in \texttt{Accepted}\} \subseteq I_r$$

holds for an exam execution E that considers any size n of the relation `Accepted` and any number m of registered candidates.

Let $testUR_k(\cdot)$ be the verifiability-test applied to an exam execution that has k accepted tests; let $testUR_k(\cdot) \rightarrow^* OK$ denote the verifiability-test that outputs `OK` (true) after some steps; let E be an exam execution that has m registered candidates and n accepted tests; let E_j be a version of E that only considers the j^{th} accepted test, which is submitted by the candidate i_j. Since ProVerif proves that the verifiability-test is complete and sound for one accepted test and any number of registered candidates, it follows that for soundness we have

$$\forall 1 \leq j \leq n : testUR_1(E_j) \rightarrow^* OK \Rightarrow i_j \in I_r$$

and for completeness we have

$$\forall 1 \leq j \leq n : i_j \in I_r \Rightarrow testUR_1(E_j) \rightarrow^* OK.$$

The verifiability-test $testUR_n(E)$ checks if each of the accepted tests received on channels `bba1`, ..., `bban` was submitted by one of the candidates given on channels `bbn1`, ..., `bbnn`. The verifiability-test $\forall 1 \leq j \leq n : testUR_1(E_j)$ checks if the j^{th} accepted test received on the channel `bbaj` was submitted by one of the candidates given on the channels `bbn1`, ..., `bbnn`.

```
let testUR(pkN, pkA, ch1,...,chn, bbn1,...,bbnm, bba1,...,bban)=
 in(bbn1, (pseudo_C1, hc, r, NET_sign1));
 ...
 in(bbnm, (pseudo_Cm, hc, r, NET_signm));

 in(bba1, receipt1);
 ...
 in(bban, receiptn);

 in(ch1, (rcoin1, EA_sign_rcpt1));
 ...
 in(chn, (rcoinn, EA_sign_rcptn));

 let (quest1, answ1, pseudo_C'1) =
 checksign(EA_sign_rcpt1, pkA) in
 ...
 let (questn, answn, pseudo_C'n) =
 checksign(EA_sign_rcptn, pkA) in

 (* If the pseudonym on the BB is signed by the NET, *)
 (* it is considered as part of the relation Accepted. *)
 if (pseudo_C1, hc, r)=checksign(NET_sign1, pkN) && r=C &&
    pseudo_C1=pseudo_C'1
    ||...||
    (pseudo_C1, hc, r)=checksign(NET_sign1, pkN) && r=C &&
    pseudo_C1=pseudo_C'n
    &&...&&
    (pseudo_Cm, hc, r)=checksign(NET_signm, pkN) && r=C &&
    pseudo_Cm=pseudo_C'1
    ||...||
    (pseudo_Cm, hc, r)=checksign(NET_signm, pkN) && r=C &&
    pseudo_Cm=pseudo_C'n  then
  if receipt1=int_encrypt(((quest1, answ1, pseudo_C'1),
                           EA_sign_rcpt1), pseudo_C1, rcoin1)
     &&...&&
     receiptn=int_encrypt(((questn, answn, pseudo_C'n),
                           EA_sign_rcptn), pseudo_Cm, rcoinn)
  then event OK
  else event KO
 else event KO.
```

Figure 5.14: The Registration universal verifiability-test

We have that

$$\begin{gathered} testUR_n(E) \rightarrow^* OK \\ \Downarrow \\ \forall 1 \leq j \leq n : testUR_1(E_j) \rightarrow^* OK \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : i_j \in I_r \\ \Downarrow \\ \{i : (i,x) \in \texttt{Accepted}\} \subseteq I_r \end{gathered}$$

Thus, the verifiability-test *testUR* is sound also for the general case.

$$\begin{gathered} \{i : (i,x) \in \texttt{Accepted}\} \subseteq I_r \\ \Downarrow \\ \forall 1 \leq j \leq n : i_j \in I_r \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : testUR_1(E_j) \rightarrow^* OK \\ \Downarrow \\ testUR_n(E) \rightarrow^* OK \end{gathered}$$

Also the verifiability-test *testUR* is complete for the general case.

Marking Correctness

The verifiability-test *testUMC*, which is depicted in Figure 5.15, takes as input from the bulletin board the pseudonym of the examiner signed by the mixnet, and the mark notifications signed by the examiner and published by the exam authority. The auditor can obtain the evaluations generated by the examiner from the mark notifications. Then, the auditor checks if the mark assigned to the question of each test coincides with the mark associated to the same question on the table provided by the exam authority. Remark! intuitively ensures this requirement only if the exam authority is honest as it provides the table at the conclusion of the exam. For simplicity, we assume that one examiner marks all the tests.

ProVerif proves soundness and completeness of the verifiability-test assuming only one marked test, namely the relation `Marked` has only one entry. To prove the general case, we should consider an unbounded number of marked test. Thus, it is necessary to show that

$$testUMC(E) = true \Leftrightarrow \forall (i,x,m) \in \texttt{Marked},\ \texttt{Correct}(x) = m$$

holds for an exam execution E that considers any size n of the relation `Marked`.

Let $MC_k(\cdot)$ be the verifiability-test applied to an exam execution that has k marked tests; let $MC_k(\cdot) \rightarrow^* OK$ denote the verifiability-test that outputs `OK` (true) after some steps; let E be an exam execution that has n marked tests; let E_j be a version of E that only considers the j^{th} marked test, which was submitted by the candidate i_j and evaluated with the mark m_j, namely $(i_j, (x_j), m_j) \in \texttt{Marked}$. Since ProVerif proves that the verifiability-test is complete and sound for one marked test, it follows that for soundness we have

$$\forall 1 \leq j \leq n : MC_1(E_j) \rightarrow^* OK \Rightarrow \texttt{Correct}(x_j) = m_j$$

```
let testUMC (pkN, bbn, ch1,...,chn) =
 in(bbn, (pseudo_E, he, r, spseE));
 in(ch1, sma1);
 ...
 in(chn, sman);

 let ((ques1, ans1, pseudo_C1), sca1, mark1) =
     checksign(sma1, pseudo_E) in
 ...
 let ((quesn, ansn, pseudo_Cn), scan, markn) =
     checksign(sman, pseudo_E) in
 get correct_ans(ques'1,ans'1,=mark1) in
 ...
 get correct_ans(ques'n,ans'n,=markn) in

 if (pseudo_E, he, r) = checksign(spseE, pkN) && r = E then
  if (ques1=ques'1 && ans'1=ans1)
     &&...&&
     (quesn=ques'n && ans'n=ansn)
  then event OK
  else event KO
 else KO.
```

Figure 5.15: The Marking Correctness universal verifiability-test

and for completeness we have

$$\forall 1 \leq j \leq n : \texttt{Correct}(x_j) = m_j \Rightarrow MC_1(E_j) \rightarrow^* OK.$$

The verifiability-test $MC_n(E)$ obtains the mark evaluations from channels `ch1`, ..., `chn`, and checks if all the tests that are contained in evaluation are marked correctly. The verifiability-test $\forall 1 \leq j \leq n : MC_1(E_j)$ checks if the j^{th} test, whose evaluation is obtained from the channel `chj`, is marked correctly. Thus, it follows that

$$\begin{array}{c} MC_n(E) \rightarrow^* OK \\ \Downarrow \\ \forall 1 \leq j \leq n : MC_1(E_j) \rightarrow^* OK \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : \texttt{Correct}(x_j) = m_j \\ \Downarrow \\ \forall (i,(q,a),m) \in \texttt{Marked},\ \texttt{Correct}(x) = m \end{array}$$

Thus, the verifiability-test *testUMC* is sound also for the general case.

$$\begin{array}{c} \forall (i,x,m) \in \texttt{Marked},\ \texttt{Correct}(x) = m \\ \Downarrow \\ \forall 1 \leq j \leq n : \texttt{Correct}(x_j) = m_j \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : MC_1(E_j) \rightarrow^* OK \\ \Downarrow \\ MC_n(E) \rightarrow^* OK \end{array}$$

Also the verifiability-test *testUMC* is complete for the general case.

Test Integrity

The verifiability-test *testUTI* (Figure 5.16) takes as input from the bulletin board the pseudonyms of the candidates signed by the mixnet and the receipts of submissions plus the mark notifications generated by the exam authority. The auditor can obtain the evaluations generated by the examiner from the mark notifications. The verifiability-test then checks if the submitted tests were marked without any modification. Similarly to Marking Correctness, we assume that one examiner marks all the tests for simplicity.

ProVerif can prove that the verifiability-test is complete and sound when one accepted test and one marked test are considered. To prove the general case that considers an unbounded number of accepted tests, it is necessary to show that

$$testUTI(E) = true \Leftrightarrow \texttt{Accepted} = \{(i,x) : (i,x,m) \in \texttt{Marked}\}$$

holds for an exam execution E that considers any size of the relations `Accepted` and `Marked`.

It can be assumed that the size of the relation `Accepted` is equal to the relation `Marked`. In fact, by looking at the bulletin board, the auditor can check that the number of the receipts of submissions coincides with the number of mark notifications.

```
let testUTI(pkN, pkA, bba1,..., bban, bbn, ch1,...,chn)=
 in(bbn, (pseudo_E, he, re, spseE));
 in(ch1,((rcoin1, sca1, pseudo_C1),(rcoinA1, smaA1, pseudo_CA1)));
 ...
 in(chn,((rcoinn, scan, pseudo_Cn),(rcoinAn, smaAn, pseudo_CAn)));
 in(bba1, (receipt1, notif1));
 ...
 in(bban, (receiptn, notifn));

 let (quest1, answ1, pseudo_C'1) = checksign(sca1, pkA) in
 ...
 let (questn, answn, pseudo_C'n) = checksign(scan, pkA) in
 let ((quest'1, answ'1, pseudo_C''1), sca'1, mark1) =
     checksign(smaA1, pseudo_E) in
 ...
 let ((quest'n, answ'n, pseudo_C''n), sca'n, markn) =
     checksign(smaAn, pseudo_E) in

 if (receipt1=int_encrypt(((quest1, answ1, pseudo_C'1), sca1),
                          pseudo_C1, rcoin1) &&
     notif1=int_encrypt((((quest'1, answ'1, pseudo_C''1), sca'1,
                         mark1), smaA1), pseudo_CA1, rcoinA1) &&
     sca'1=sca1)
     &&...&&
    (receiptn=int_encrypt(((questn, answn, pseudo_C'n), scan),
                          pseudo_Cn, rcoinn)&&
     notifn=int_encrypt((((quest'n, answ'n, pseudo_C''n), sca'n,
                         markn), smaAn), pseudo_CAn, rcoinAn) &&
     sca'n=scan)
 then
  if (pseudo_C1=pseudo_CA1 && pseudo_CA1=pseudo_C'1 &&
      pseudo_C'1=pseudo_C''1 && quest1=quest'1 && answ1=answ'1)
     &&...&&
     (pseudo_Cn=pseudo_CAn && pseudo_CAn=pseudo_C'n &&
      pseudo_C'n=pseudo_C''n && questn=quest'n && answn=answ'n)
  then event OK
  else KO
 else KO.
```

Figure 5.16: The Test Integrity universal verifiability-test

Let $testUTI_k(\cdot)$ be the Test Integrity verifiability-test applied to an exam execution that has k accepted tests and k marked tests; let $testUTI_k(\cdot) \rightarrow^* OK$ denote the verifiability-test that outputs `OK` (true) after some steps; let E be an exam execution that has n accepted tests and n marked tests; let us assume that the tests are marked in the same order as they were accepted; let E_j be a version of E that only considers the j^{th} accepted test x_j submitted by the candidate i_j, and the j^{th} marked test x'_j associated to the candidate i'_j.

Since ProVerif proves that the verifiability-test is complete and sound for one accepted test and one marked test, it follows that for soundness we have

$$\forall 1 \leq j \leq n : testUTI_1(E_j) \rightarrow^* OK \Rightarrow (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j))$$

and for completeness we have

$$\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j)) \Rightarrow testUTI_1(E_j) \rightarrow^* OK.$$

The verifiability-test $testUTI_n(E)$ checks if each pair of accepted and marked tests obtained from channels `bba1`, ..., `bban` has the same pseudonym, question, and answer. Similarly, the verifiability-test $\forall 1 \leq j \leq n : testUTI_1(E_j)$ checks if the j^{th} accepted and marked tests obtained from the channel `bbaj` are identical. Thus, it follows that

$$\begin{array}{c} testUTI_n(E) \rightarrow^* OK \\ \Downarrow \\ \forall 1 \leq j \leq n : testUTI_1(E_j) \rightarrow^* OK \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : (i_j, x_j) = (i'_j, x'_j) \\ \Downarrow \\ \texttt{Accepted} = \{(i, x) : (i, x, m) \in \texttt{Marked}\} \end{array}$$

Thus, the verifiability-test $testUTI$ is sound also for the general case.

$$\begin{array}{c} \texttt{Accepted} = \{(i, x) : (i, x, m) \in \texttt{Marked}\} \\ \Downarrow \\ \forall 1 \leq j \leq n : (i_j, x_j) = (i'_j, x'_j) \\ \Downarrow \\ \forall 1 \leq j \leq n : testUTI_1(E_j) \rightarrow^* OK \\ \Downarrow \\ testUTI_n(E) \rightarrow^* OK \end{array}$$

Also the verifiability-test $testUTI$ is complete for the general case.

Test Markedness

Since Remark! is test integrity universally verifiable, it is also test markedness universally verifiable. The proof strategy is the same outlined above for Test Integrity. However, it is not necessary to assume that the size of the relation `Accepted` is equal to the relation `Marked`, since Test Markedness does not require strict equality of the two multisets.

```
let testUMI (bbn, bba1,...,bban) =
 in(bbn, (pseudo_E, he, re, spseE));
 in(bba1, (notif1,rcoin1, sma1));
 ...
 in(bban, (notifn,rcoinn, sman));

 let ((quest1, answ1, pseudo_C1), sca'1, mark1) =
     checksign(sma1, pseudo_E) in
 ...
 let ((questn, answn, pseudo_Cn), sca'n, markn) =
     checksign(sman, pseudo_E) in

 if notif1=int_encrypt((((quest1, answ1, pseudo_C1),sca'1, mark1),
                        sma1), pseudo_C1, rcoin1)
    &&...&&
    notifn=int_encrypt((((questn, answn, pseudo_Cn),sca'n, markn),
                        sman), pseudo_Cn, rcoinn)
 then event OK
 else event KO.
```

Figure 5.17: The Mark Integrity universal verifiability-test

Mark Integrity

The verifiability-test *testUMI*, which is in Figure 5.17, is fed with mark notifications posted on the bulletin board by the exam authority. The auditor obtains the evaluations generated by the examiner from the mark notifications, and checks if the marks that the exam authority assigned to the candidates coincide with the marks that the examiner assigned to the candidates' tests. Also for this requirement we assume one examiner who marks all the tests.

In the case that the relations `Assigned` and `Marked` contain each one entry, ProVerif proves that the verifiability-test Mark Integrity is complete and sound. The general case, which considers an unbounded number of entries, consists on proving that

$$testUMI(E) = true \Leftrightarrow \texttt{Assigned} = \{(i, m) : (i, x, m) \in \texttt{Marked}\}$$

holds for an exam execution E that considers any size of the relations `Assigned` and `Marked`.

Similarly to Test Integrity, it can be assumed that the size of the relation `Assigned` is equal to the relation `Marked`, as the auditor can check such equality by looking at the bulletin board.

Let $testUMI_k(\cdot)$ be the Mark Integrity verifiability-test applied to an exam execution that has k marks assigned to the candidates and k marks associated to the tests; let $testUMI_k(\cdot) \rightarrow^* OK$ denote the verifiability-test that outputs `OK` (true) after some steps; let E be an exam execution that has n marks assigned to the candidates and n marks associated to the candidates' tests; let us assume that the tests are assigned to the candidates in the same order as they were

Requirement	Soundness	Completeness
Registration	✓	✓(all)
Marking Correctness	✓ (EA)	✓(all)
Test Integrity	✓	✓(all)
Test Markedness	✓	✓(all)
Mark Integrity	✓	✓(all)

Table 5.4: Summary of the analysis of Remark! for U.V. requirements

marked; let E_j be a version of E that only considers the j^{th} mark m_j assigned to the candidate i_j, and the j^{th} mark m'_j associated to the test of candidate j'_j.

Since ProVerif proves that the verifiability-test is complete and sound for one entry, it follows that for soundness we have

$$\forall 1 \leq j \leq n : testUMI_1(E_j) \rightarrow^* OK \Rightarrow (i_j, m_j) = (i'_j, m'_j)$$

and for completeness we have

$$\forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \Rightarrow testUMI_1(E_j) \rightarrow^* OK.$$

The verifiability-test $testUMI_n(E)$ receives from the channels `bba1`, ..., `bban` the notifications of the exam authority, and checks if the pseudonyms and marks obtained from the notifications coincide with the ones obtained from the evaluations of the examiner. Similarly, the verifiability-test $\forall 1 \leq j \leq n : testUMI_1(E_j)$ checks if the j^{th} pseudonym and mark obtained from the evaluation and notification on channel `bbaj` are identical. Thus, it follows that

$$\begin{gathered} testUMI_n(E) \rightarrow^* OK \\ \Downarrow \\ \forall 1 \leq j \leq n : testUMI_1(E_j) \rightarrow^* OK \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \\ \Downarrow \\ \{(i, m) : (i, x, m) \in \texttt{Marked}\} = \texttt{Assigned} \end{gathered}$$

Thus, the verifiability-test *testUMI* is sound also for the general case.

$$\begin{gathered} \{(i, m) : (i, x, m) \in \texttt{Marked}\} = \texttt{Assigned} \\ \Downarrow \\ \forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : testUMI_1(E_j) \rightarrow^* OK \\ \Downarrow \\ testUMI_n(E) \rightarrow^* OK \end{gathered}$$

Thus, the verifiability-test *testUMI* is complete also for the general case.

Table 5.4 summarises the results of the universal verifiability analysis of Remark! and reports the roles required to be honest.

5.6 Conclusion

This chapter presents Remark!, a protocol for Internet-based exam that guarantees authentication, privacy, and verifiability with minimal trust assumptions. Remark! meets its requirements in most of the cases by assuming only one honest server among the servers that compose the exponentiation mixnet. According to each requirement, Remark! can resist against collusion of candidate and exam authority (e.g., Anonymous Examiner), exam authority and examiner (e.g., Anonymous Marking), or candidate and examiner (e.g., Question Indistinguishability) without the presence of a trusted third party.

A formal analysis in ProVerif confirms that Remark! ensures all the authentication and privacy requirements proposed in chapter 3. Notably, thanks to this formal analysis, we found and solved an issue on an earlier version of the protocol.

Remark! proves to be fully verifiable, according the individual and universal verifiability definitions proposed in chapter 4. ProVerif automatically proves all the individual verifiability requirements. Assuming an honest mixnet, all the requirements but Question Validity and Marking Correctness can be proved assuming corrupted candidates, examiners, and exam authority. Question Validity and Marking Correctness still require an honest exam authority. Concerning the universal verifiability requirements, ProVerif cannot deal with the general cases, thus we completed the analysis with manual proofs. It turns out that Remark! ensures all the requirements but Marking Correctness assuming corrupted candidates, examiners, and exam authority. Also in this case, Marking Correctness can be proved assuming an honest exam authority. However, it is also assumed that the exam authority provides the auditor with some additional data at the conclusion of the exam, since all the messages posted on the bulletin board are encrypted.

Chapter 6

Computer-assisted Exam Protocols

According to Definition 2 proposed in chapter 2, a protocol belongs to the category of computer-assisted exams if at least one if its phases receives some level of assistance from computers. We argued that the levels of detail and abstraction of the protocol specification determine whether a protocol belongs to traditional or to computer-assisted exams.

In this chapter, we focus on some specifications of computer-assisted exam protocols that share traditional testing, namely testing takes place by pen and paper. These protocols however have different functional requirements and threat models: one considers local tasks, such as notification of marks, and no TTP; some others consider remote tasks, such as remote registration, but assume TTP; one achieves remote tasks without TTP.

In a way, Remark! already achieves remote registration and remote notification with minimal reliance on trusted parties. As Remark! belongs to the class of Internet-based exams, it requires candidate and exam authority to use computers at testing in order to sign and encrypt the tests. Therefore, testing cannot take place by pen and paper. Moreover, Remark! assumes at least one honest mix server. As we shall see later, we propose a computer-assisted exam protocol that ensures the same authentication and privacy requirements of Remark! though relying neither on mixnet nor on TTP.

Either testing is carried out traditionally or remotely is a key aspect for security. Remote testing is supported with computers, which intuitively introduce more security risks, but facilitate the design of the other remote phases. In contrast, traditional testing is less risky but complicates such remote design. The major security risks introduced by remote testing are due to remote invigilation and computer devices required at testing. Normally, it is better not to allow such devices at testing because they can promote candidate cheating. For instance, Migicovsky *et al.* [MDRH14] have recently shown how to outsmart invigilation using a smart watch. The difference between traditional testing and remote testing, namely between computer-assisted exams and Internet-based exams, finds its analogue in voting: the former is comparable to paper-based electronic voting systems, while the latter is the analogous of Internet-voting systems.

In this chapter, we focus on a family of computer-assisted exam protocols called *WATA*. The original versions of WATA progressively introduce more computer assistance in their design still keeping traditional testing and assuming TTP. We then propose a novel protocol, WATA IV, which includes the lightweight participation of a TTP, opens up also for computer-based exams, and ensures more security requirements despite a stronger threat model. WATA IV is further reconceived to completely remove the TTP. The underlying idea is to combine oblivious transfer and visual cryptography to allow candidate and examiner to jointly generate a pseudonym that anonymises the candidate's test. The pseudonym is revealed only to the candidate at the beginning of testing. We analyse the protocol formally in ProVerif and prove that it satisfies all the stated security requirements.

Outline of the chapter. Section 6.1 discusses the related work about secure protocols for computer-assisted exams. Section 6.2 describes the original WATA II & III protocols and their informal analyses. Section 6.3 introduces WATA IV according the four phases of an exam and points up the novelties respect to the original WATA schemes. Section 6.4 contains a new exam protocol that redesigns parts of WATA IV and removes the TTP. This section also introduces the formalisation of dispute resolution, and provides the formal analysis of the new exam protocol in ProVerif. Section 6.5 discusses future work and ends the chapter.

6.1 Related Work

Nowadays, most of the exams employed in public competitions are computer-assisted or even computer-based. ETS and Pearson Vue develop various Computer-assisted exams for skill and professional certifications [ETS15, Pea15]. The European Union adopts computer-based exams for the selection of EU personnel [Off13]. The specification of such exams is not publicly available, and their security fully relies on the developers, who have the prominent role of TTP during the exam execution. This choice has not prevented frauds on the administered exams [Wat14].

Different exam protocols have been proposed to ensure anonymous marking. INFOSAFE [INF15] is an anonymous marking system for computer-assisted exam with traditional testing, and is adopted in university exams. Candidates write down their personal details on top of a tamper-evident paper, and hide them with a flap which is bent and glued over. After marking, the personal details are disclosed by tearing off the flap. Systems following a "double envelope" strategy, often used in public tenders, make use of two envelops to separate the identification details from the offers. The personal details are assumed to be read after marking. Many European universities, such as Dublin City University and University of Sheffield, use their own anonymous marking systems [Uni15, oS15]. So do top USA academies, such as Stanford and Harvard Law School [Sch15, Lev04]. The latter relies on the Blind Grading Number system which assigns candidates with numerical pseudonyms until the marking period ends. Nemo Scan [Neo15] uses a patented anonymity paper cover [Mou09] consisting of two parts: one with the covered candidate details, the other with a

section where to type the marks. At notification, a scanner with a proprietary software reads the paper with the candidate details and assigns her the mark.

All the systems outlined above assume a trusted authority to ensure Anonymous Marking. Moreover, it is not clear how such systems scale up to other security requirements. In this chapter, we discuss how to progressively remove trusted authorities from the design of the protocols, and consider the exam authority corrupted to various extents.

6.2 The WATA Exam Protocols

The acronym WATA stands for Written Authenticated Though Anonymous exams, and refer to a family of exams originally developed at the University of Catania. Historically, WATA exams have two main goals. The first goal is to mechanise the double envelope technique in a software. The second goal is to ensure authentication and anonymity despite a corrupted examiner.

The first two versions of the system are conceptually identical and only differ in the implementations: WATA I was written in Visual Basic and was only available for Microsoft Windows; WATA II [BCR10] was implemented in Java, hence more efficient and portable. In this chapter, we only consider the second version.

WATA III [BCCKR11] redesigns completely the exam system to offer remote management and remote notification, features not available in the previous versions. However, the new design introduces a TTP that participates in all the phases of the exam. We first propose WATA IV, a new version that minimises the involvement of such TTP, and then we show how to remove it completely from the design. Every version considers candidates free of long-term public keys, and contemplates either traditional testing or computer-based exams, but not remote testing.

The WATA exams were originally incepted as software rather than protocols. In the first two versions, the software ran locally into the examiner's computer, while in WATA III the software ran into a remote machine. We provide a different prospective of the WATA exams by originally describing them as protocols.

Notification Request Authentication

The WATA exam was originally conceived for university exams, and in some universities nowadays candidates can take the exam up to a fixed number of times. However, if the candidate withdraws, it is not counted towards the number of attempts. Other universities have a policy that does not allow the candidate to resit a failed exam the next session, unless the candidate withdraws from the exam before notification. Thus, WATA exams consider the additional requirement of *Notification Request Authentication*. It says that a mark should be associated with the candidate only if she requests to learn her mark.

To formalise this requirement in the applied π-calculus, we need to define two new events that extend the list proposed in chapter 3.

- $\texttt{requested}\langle id\_c, id\_test\rangle$ means that the candidate id_c accepts to learn the mark associated to the test id_test. The event is inserted into the process of the candidate at the location where the request is sent to the notifier.

- $\texttt{stored}\langle id\_c, mark\rangle$ means that the authority officially considers the candidate id_c associated with $mark$. The event is inserted into the process of the authority at the location where it registers the mark to the candidate.

The requirement can be specified as follows:

Definition 42 (Notification Request Authentication) *An exam protocol ensures* Notification Request Authentication *if for every exam process EP*

$$\texttt{stored}\langle id\_c, mark\rangle \;\rightsquigarrow\; inj\texttt{requested}\langle id\_c, id\_test\rangle$$

on every execution trace.

We use the specification above to formally analyse the protocol described in Section 6.4.

6.2.1 WATA II

WATA II considers the roles of examiner, invigilator, and candidate. The examiner, in addition to the usual tasks assigned to its role, runs tasks normally ascribed to other authorities, such as the question committee, the recorder and the notifier. The invigilator distributes the tests to the candidates, and collects them at the end of testing. Every phase of the exam is executed locally, and testing takes place traditionally by pen and paper.

The examiner maintains data in three tables: the history table DB_h records the performances of the candidate over the past exam; the mark table DB_m stores the mark assigned to each test; The question table DB_q stores the questions.

In the following, we describe the protocol and refer to the message sequence chart depicted in Figure 6.1.

Preparation

The examiner randomly extracts a list of questions *quest* from the question table, and generates a random test identifier *id_test* of predetermined size n using the alphabet Σ. The test identifier is stored in the mark table next to an empty mark. The examiner then prints out the `test`, which contains the following information: the test identifier *id_test*, an authentication form *auth_form*, another occurrence of the test identifier, the questions, and a form for the answers *answ_form*. To facilitate the mechanical reading, both occurrences of the test identifier are encoded as a barcode. The test has a precise layout, notably with test identifier and authentication form framed at the top-left corner of the sheet through a dotted line; this can be seen in Figure 6.2. The examiner signs inside this frame across test identifier and authentication form and possibly reinforces the signature with the stamp of the exam organisation. It is assumed that this association is tamper-proof. This produces $\texttt{test}_{\texttt{signed}}$, which the examiner hands to the invigilator (step 1). This phase is repeated as many times as the number of registered candidates, so that the invigilator gets a pile of tests pre-signed by the examiner.

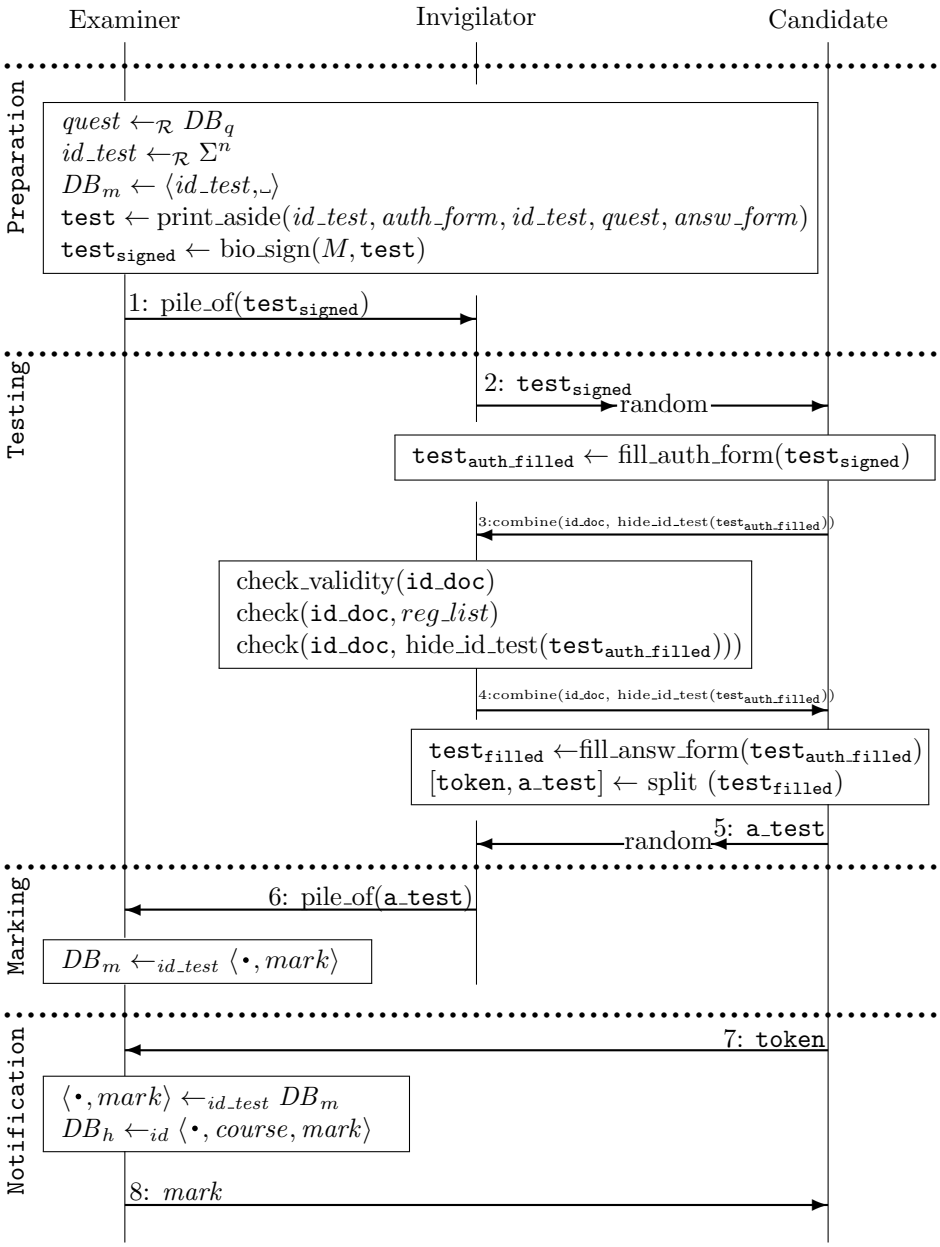

Figure 6.1: The WATA II exam protocol

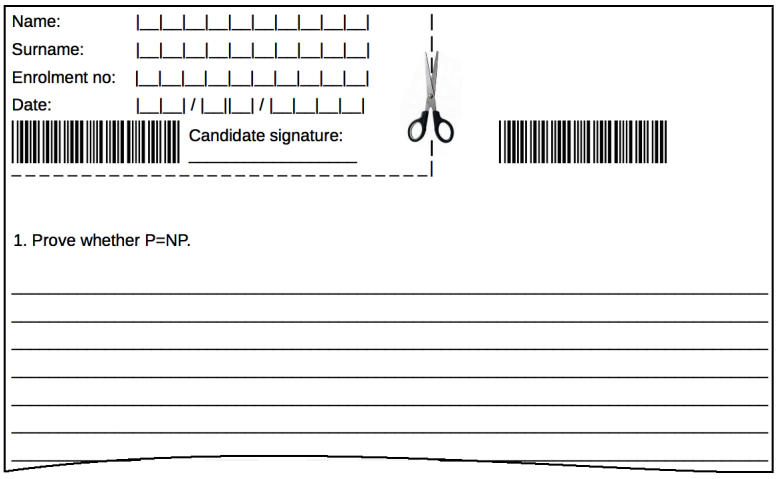

Figure 6.2: Fragment of a test in WATA II

Testing

The invigilator leaves the pile of tests on a desk at the exam venue. Then, the candidate picks a test randomly (step 2), and is assigned a seat. At her seat, the candidate fills out the authentication form with her personal details, and temporarily hides the test identifier prior to hand the test combined with her identity document to the invigilator for authentication (step 3). The hiding could be achieved, for example, by folding the top corners in. In so doing, the invigilator cannot learn the test identifier while authenticating the candidate. The invigilator checks whether the identity document is the candidate's valid one, whether the candidate identity matches an entry in the list of registered candidates and the details written on the authentication form. If so, the invigilator hands identity document and test back to the candidate (step 4), who can fill out the answer section.

When the testing time is over, the candidate tears the test in two pieces of papers of different sizes. The smaller one, which we term *token*, contains the filled authentication form and the test identifier. The larger piece of paper, which we term *a_test*, contains questions, test identifier, and answers. The candidate keeps the *token*, and leaves the anonymous test in a random position through the current pile of tests (step 5).

Marking

The invigilator collects the pile of anonymous tests and distributes them to the examiners (step 6). It also removes the records of the mark table that refer to undistributed tests.

The examiner evaluates the answers and assigns a mark to the anonymous test. The examiner then scans the barcode to get the corresponding test identifier, and enters the mark in the mark table, precisely in the record identified by the test identifier.

Notification

The candidate who wants to know her mark brings her token at the venue announced to host the notification. There the candidate hands the token to the examiner (step 7), who checks signature and personal details, and scans the barcode to get the corresponding test identifier. The examiner finds the record identified by the test identifier on the mark table, and obtains the corresponding mark. Finally, the examiner stores the mark into the history table, and notifies the mark to the candidate (step 8).

Discussion

Except for the preparation of the tests, the presence of computers in WATA II is minimal. Most of the tasks are run by humans, and the security of the protocol mostly relies on the physical properties of paper.

We consider the requirements proposed in chapter 3. It can be observed that WATA II trivially ensures Test Authenticity but not Anonymous Examiner because the protocol considers only one examiner. Candidate Authorisation is met because the invigilator authorises the candidate to take the exam only if the personal details reported on the identity document match an entry in the list of eligible candidates. Also Answer Authenticity is met because the invigilator verifies that the candidate wrote the personal details on the authentication form. Moreover, the examiner's signature on the tests ensures their authenticity. It follows that WATA II also ensures Test Origin Authentication. Mark Authenticity is met because the examiner inserts the mark into the mark table, exactly in the record identified by the random test identifier reported into the test. At notification, the examiner notifies the candidate with this mark, which is also stored in the history table. However, a malicious examiner may tell a different mark after the candidate hands him the token. The novel requirement of Notification Request Authentication is met because only the candidate holds the token. A malicious examiner cannot generate a forged token because it would need to be signed by the candidate. In fact, the candidate is the sole entity who can establish the link between her identity and her test.

Concerning privacy requirements, WATA II guarantees Question Indistinguishability provided that the candidate does not collude either with the examiner or the invigilator. In fact, the examiner hands the tests to the invigilator prior testing. However, if the questions that appear into a test are randomly chosen, it becomes harder for a candidate to learn which questions she will be assigned. Anonymous Marking is met because only the candidate knows the test identifier associated to her. Moreover, the candidate submits the test in a random position of the pile of anonymous tests. WATA II ensures Mark Privacy because the notification is face-to-face. The examiner notifies the mark to the corresponding candidate after a successful authentication, and only if she hands a valid token. Since Mark Privacy is met, it follows that also Mark Anonymity is met.

Although WATA II ensures authentication and privacy without TTP, it has the major limitation that notification requires candidates to meet in person the examiner.

6.2.2 WATA III

WATA III allows remote notification and involves a major level of computer assistance than the previous version. The protocol considers the participation of the *WATA Server* in addition to candidate, examiner, and invigilator roles. The WATA Server runs most of the tasks of preparation and notification, while the tasks of the examiner are now limited to the marking phase.

The WATA Server maintains data in four tables. The history, mark, and question tables have the same functionalities of the previous protocol. WATA III introduces the *sharec* table DB_c that stores the partial information about the test identifier, which now is called *pseudonym*.

The pseudonym is associated also to the candidate rather than only to the test. The idea is to split the pseudonym in two shares, print them on the test, and give one to the candidate and the other to the examiner. The latter can associate a test to its author only if the candidate reveals its share.

WATA III assumes that a list of registered candidates for the exam is available to the invigilator, and secure TLS communications between the WATA Server and the other principals. The WATA Server authenticates invigilator and candidate via login and password. Every communication between invigilator and candidate is face-to-face, while communications to the WATA Server are always remote. Remote communications are highlighted with dashed lines in the message sequence chart in Figure 6.3.

Remark. We found a security issue in the original specification of WATA III [BCCKR11]. In a nutshell, the specification allowed a corrupted candidate to be assigned with the mark of the test submitted by another candidate, hence violating Mark Authenticity. The corrupted candidate could generate a fake pseudonym such that at notification she could convince the examiner that the pseudonym should associated with her identity. This was possible because the original specification contemplates a weak generation of the pseudonym that reveals key information. We fix this issue by hide such information via hashing. In the remainder, we only describe the fixed version.

Preparation

At exam venue, the candidate approaches the invigilator's desk and hands her identity document `id_doc` (step 1). The invigilator checks whether the personal details of the candidate *id* appears in the registered candidate list, and if so, logs in the WATA Server and sends *id* via secure channel (step 2). The WATA Server randomly extracts the questions from the question table. Then, it generates a random value *sharek*, whose length matches the *id* augmented with some randomness *id_rnd*. Thus, it generates (*sharec*) that is the result of the one-time pad of *sharek* with *id_rnd*. The WATA Server stores *sharec* in the database and finally generates the layout of the test. The *sharek* is placed on the top left of the printout, while the hashed version of *sharec* is placed on the top right of the printout, close to the answer section *answ_form*. Both *sharek* and hashed *sharec* are represented in the form of barcode. The test thus consists of two parts: the *token*, which contains *sharek* and the candidate's personal details *id*; the anonymous test *a_test*, which contains the hash of *sharec*, questions, and answer section.

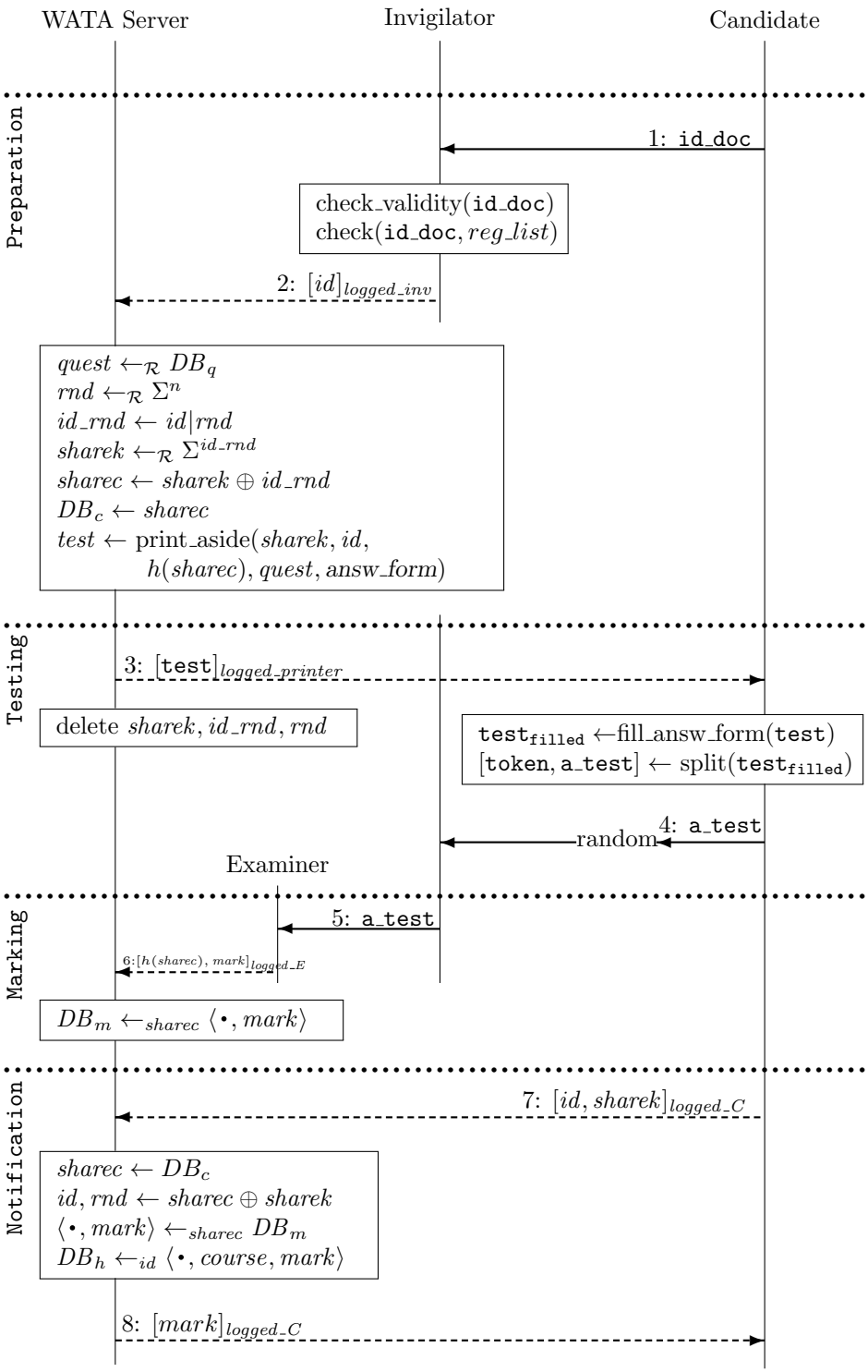

Figure 6.3: The WATA III exam protocol

Testing

The WATA Server sends the test via a secure channel to a printer, which is available at the exam venue. The candidate approaches the printer and takes its `test` (step 3). Then, the WATA Server deletes all the data it used but the one stored in the tables, namely it removes *sharek*, *id_rnd*, and the randomness.

At her sit, the candidate fills out the answer section with the answers. When the time is over, the candidate splits the test, and takes the `token` at home while inserts the anonymous test `a_test` into a random position of the pile of anonymous tests (step 4).

Marking

The invigilator collects the pile of anonymous tests and hands them to the examiner (step 5), who evaluates the answers and assigns a mark to each anonymous test. For each test, the examiner scans the barcode and gets the corresponding hash of *sharec*. The examiner then logs into WATA Server and uploads the pair of hashed *sharec* and mark via a secure channel (step 6). The WATA Server can find the corresponding *sharec* by hashing each entry of DB_c. In so doing, it can store the mark in the entry identified by *sharec* in the mark table.

Notification

The candidate who wants to know her mark scans the barcode printed in the token and gets *sharek*, which she sends to the WATA Server via a secure channel, after she logged in (step 7). The WATA Server XOR-es *sharek* with each *sharec* stored in the database until it decrypts a valid *id* concatenated with some randomness. It then retrieves the record identified by the *sharec* from the mark table, and obtains the corresponding mark. Finally, the WATA Server stores the mark into the history table, and notifies it to the candidate (step 8).

Discussion

It is clear that WATA III cannot be deemed secure assuming a corrupted WATA Server. This role is ubiquitous in the design and is in charge of the critical steps. Therefore, the WATA Server should be considered as an honest-but-curious role, which follows the protocol honestly but tries to learn as much as possible.

A critical part of the protocol is the deletion of data performed by the WATA Server. Although this practice is found in other protocols [EKOT14], it may be impractical to force a party to delete data. However, it can be still possible to verify that the party actually deletes the data [HCZ15].

Concerning the authentication requirements, WATA III ensures Candidate Authorisation since the invigilator authorises the candidate to take the exam only if the candidate's personal details reported on the valid identity document match an entry in the list of eligible candidates. The invigilator has to ensure that the correct candidate takes the test generated for her from the printer. This avoids that corrupted candidates swap their tests before sitting for the exam. In fact, the protocol does not require that the candidate writes her personal details down into the test as provided for the previous version. Thus, the invigilator does not need to check the test once the candidate sits for the exam. Answer Authenticity is met because candidates are invigilated. If a

corrupted candidate introduces an illegal test, she will not be able to receive a mark because the examiner would upload a forged hash of *sharec* that the WATA Server could not retrieve in the database. It follows that WATA III ensures Test Origin Authentication. Similarly to WATA II, Test Authenticity is met but not Anonymous Examiner since the protocol considers only one examiner. Mark Authenticity is met because both candidate and examiner know only the hash of *sharec* until notification. WATA III ensures Notification request authentication because only the candidate knows the *sharek* after testing.

Concerning privacy requirements, Question Indistinguishability is met because the WATA Server generates the test with the questions. Each test is printed at testing and taken by the candidate directly, so the invigilator learns the question only when testing concludes. Although the WATA Server deletes some data at testing, we observe that it can violate Anonymous Marking. At preparation, a corrupted WATA Server can associate the candidate *id* with the corresponding *sharec*. At testing, the WATA Server receives from the examiner the mark associated with the hash of the *sharec*. Therefore, the WATA Server can learn the author of a test without the knowledge of *sharek*. Note that this attack is possible even considering an honest-but-curious threat model because it does not require that the WATA Server deviates from the protocol, but resorts solely on the knowledge of the WATA Server. Finally, Mark Privacy is met because the WATA Server notifies the mark to the candidate only, after receiving a correct *sharek*. It follows that WATA III ensures Mark Anonymity as well.

WATA III allows for remote notification but assigns most of its critical tasks to a TTP. It turns out that considering an honest-but-curious WATA Server, WATA III fails to ensure Anonymous Marking.

6.3 WATA IV

In this section, we advance WATA IV, a new exam protocol that overcomes the limitations of WATA III. We design the protocol without the ubiquitous WATA Server and introduce the *anonymiser* role, whose participation is confined to preparation only. Similar to the WATA Server, the anonymiser is honest-but-curious but its duties are drastically reduced. In WATA IV most of the critical tasks are run by a possibly corrupted examiner. Moreover, WATA IV opens for remote preparation and requires no printers at testing. We anticipate that WATA IV meets the same security requirements as WATA III does, though augmented with Anonymous Marking and the two individual verifiability requirements of Mark Integrity and Mark Notification Integrity.

The main novelty of the design of WATA IV is the use visual cryptography. The idea consists of encoding the pseudonym into two visual cryptographic shares: one share is given to the candidate, and the other is given to the examiner. Neither the candidate nor the examiner knows the pseudonym until they meet at testing, when the candidate learns it by overlapping the examiner's share with hers.

Thanks to visual cryptography, the candidate can do a cryptographic operation at testing without the assistance of any computer device. In the following, we briefly discuss visual cryptography and commitment scheme, namely the cryptographic primitives used in WATA IV.

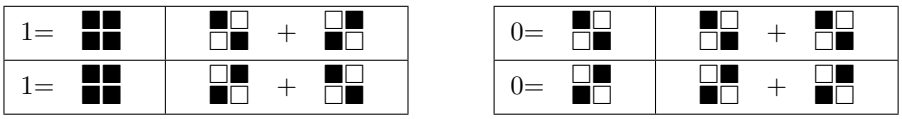

Figure 6.4: Representation of bits 0 and 1 using visual cryptography

Visual Cryptography

It is a secret sharing scheme, devised by Naor and Shamir [NS95], that allows a visual decryption of a ciphertext. A secret image is "encrypted" by splitting it into a number of image *shares*. The basic version of the scheme is the 2-out-of-2 secret sharing system, in which a secret image is split into two shares $share_A$ and $share_B$. The shares are printed on transparency sheets, which reveal the secret image when the shares are overlapped. This scheme is information-theoretic secure, namely each share leaks no information about the secret image. In fact, it emulates the XOR operation though the visual decryption is actually equivalent to the OR operation. The scheme is information-theoretic secure because either a black or a white pixel, mapped respectively to 0 and 1, can originate by any of the sub-pixels shown in Figure 6.4.

Many schemes for visual cryptography have been proposed over the years. Although we consider the Naor and Shamir scheme for WATA IV, we conjecture that other visual scheme can be used as well, but with different security guarantees.

Commitment Scheme

A commitment scheme is used to bind a committer to a secret value. The committer publishes a commitment that hides the value, which remains secret until he reveals it. Should the committer reveal a different value, this would be noticed because it cannot be mapped to the published commitment. WATA IV uses the Pedersen commitment scheme [Ped92], which guarantees unconditional hiding, namely the value remains secret despite a computational unbounded attacker. The scheme consists of the algorithms of *commitment*, in which the value is chosen, hidden, and bound to the committer, and of *disclosure*, in which the value is publicly revealed. The commitment algorithm takes in two given public generators $g, h \in \mathbb{G}_q$, the secret value v, and a random value $r \in_{\mathcal{R}} \mathbb{Z}_q^*$. The algorithm outputs the commitment $g^v h^r$ denoted with $C_r(v)$. The disclosure algorithm takes in the commitment $C_r(v)$, the values v and r, and outputs `true` if the commitment is correct or `false` otherwise.

WATA IV adopts the Pedersen commitment scheme at notification. The examiner generates a commitment of the mark of the candidate. Once the candidate reveals her identity to know the mark, she can verify the examiner notifies her the committed mark. This deters the examiner to notify the candidate with a mark that is different than the one assigned to candidate's test.

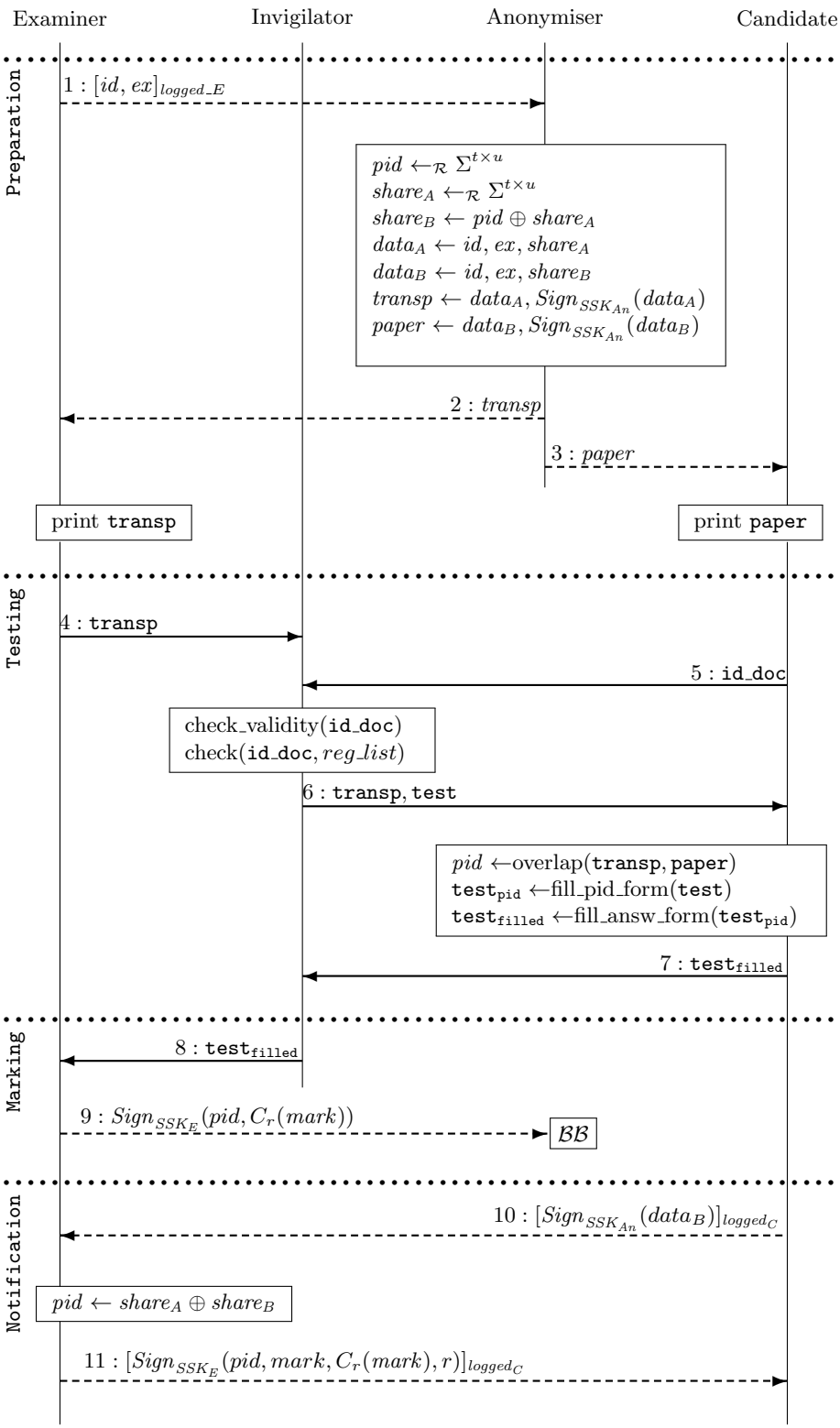

Figure 6.5: The WATA IV exam protocol

6.3.1 Description

WATA IV relies on a append-only bulletin board in which the examiner publishes the pseudonyms and the commitment of the marks. Each session is identified with a unique exam code *ex*. In the following we describe the protocol and refer to the message sequence chart depicted in Figure 6.5.

Preparation

The examiner checks the candidate eligible for the exam *ex* and, if so, enters the candidate details *id* in the dedicated list *reg_list*. After that, the examiner sends the candidate's details and the exam code to the anonymiser via a secure channel (step 1).

The anonymiser generates the pseudonym *pid* that consists of a visual representation of a random alphanumeric string, and a random visual cryptographic image, $share_A$. Then, the anonymiser generates the second visual cryptographic image, $share_B$, such that overlapping $share_A$ and $share_B$ results in the image representing of the pseudonym. Let $data_A$ denote the triplet of *id*, *ex* and $share_A$. The anonymiser signs $data_A$, and generates the corresponding signature as follows. First, the plaintexts $data_A$ and $data_B$ are encoded in *Base64* and signed with the signing key of the anonymiser SSK_{An}. Then, the signatures are encoded in *Base64* again, and included in two QR codes with the corresponding encoded plaintext. To facilitate the printing, the anonymiser includes such information in the digital versions of an A4 paper sheet, respectively *transp* and *paper*, which layout is outlined in Figure 6.6. The signatures printed on the bottom of the sheets self contain the data reported on each sheet plus the corresponding signature. The anonymiser emails *transp* to the examiner (step 2) and *paper* to the candidate (step 3). We assume that the attacker is out of control of the email infrastructure, hence he cannot learn the contents. However secure emailing techniques, such as S/MIME [RT10] or MIME over OpenPGP [EDTLR01], can be used to ensure confidentiality of the content of the emails.

For each candidate, the examiner stores the signed $data_A$ into the database, and prints each `transp` on a transparency sheet. Similarly, each candidate prints her `paper` on a common A4 paper sheet.

Testing

The examiner hands the transparency sheet to the invigilator (step 4). The candidate takes a seat at exam venue, and hands a valid identity document `id_doc` to the invigilator for authentication (step 5). The invigilator checks that the candidate is in the list of those registered for the exam. Then, the invigilator finds the transparency sheet `transp` that reports the candidate's details, and hands it to her along with a `test` (step 6). If some registered candidates fail to show up, the invigilator puts the corresponding transparency sheets and the excess tests aside.

Once the invigilator delivers the transparency sheets to all candidates, the candidate can overlap her paper sheet with the corresponding transparency sheet and can read the pseudonym. The candidate writes down the pseudonym into the test and begins to answer the questions. When the testing time is over, the candidate submits her test (step 7), and takes the paper and transparency sheets back with her. The candidate can place her test anywhere in the pile

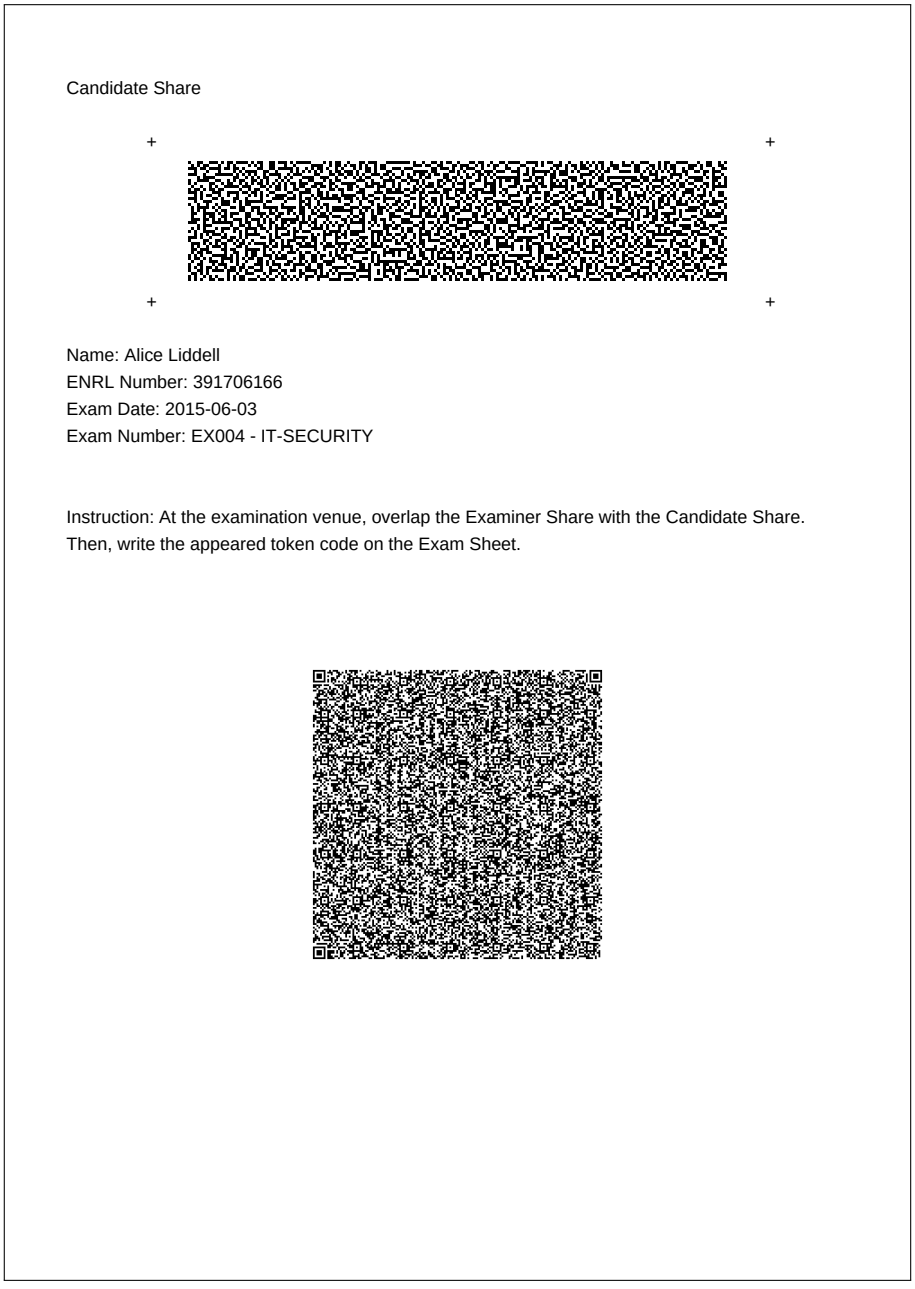

Figure 6.6: The candidate paper sheet in WATA IV

of already submitted tests. The invigilator collects the pile of tests when all candidates have submitted their tests.

Marking

The invigilator hands all the tests (step 8) and the remaining transparency sheets to the examiner. The examiner evaluates the test, generates a commitment of the *mark*, signs the pair of pseudonym and mark (*pid*, *mark*) and publishes the signature on the public append-only bulletin board (step 9). This allows the candidate to verify whether her test has been marked though ignoring the mark.

Notification

The examiner runs notification for a fixed time frame. The candidate who wants to know her mark sends the signed $data_B$ to the examiner via a secure channel (step 10). The examiner verifies the signature, and overlaps $share_B$ with its $share_A$ to get the pseudonym *pid*. Notably, this procedure can be implemented, hence requires no human involvement. The examiner thus retrieves the mark associated with the pseudonym, and finally sends to the candidate the signature of pseudonym, mark, commitment, and commitment random value (step 11).

6.3.2 Discussion

WATA IV ensures all the security requirements as WATA III does plus a few more. Concerning authentication, Test Authenticity is trivially met while Anonymous Examiner is not because the protocol assumes only one examiner. Candidate Authorisation is met because the invigilator checks that the candidate is in the list of those registered for the exam. Answer Authenticity is met because the invigilator gives to the candidate the transparency sheet that has her details. If a corrupted candidate prints a different visual crypto image on her paper sheet, she cannot read any intelligible pseudonym by overlapping the paper sheet with the transparency sheet. The same applies if any two corrupted candidates swap their paper sheets before testing. As we shall see later, a dispute resolution procedure guarantees that a corrupted candidate cannot even claim that no pseudonym appears because the examiner misprinted the transparency sheet. Still, a corrupted candidate could write a random pseudonym into the test, but at notification the candidate would not be able to send a valid $data_B$. Since WATA IV ensures Candidate Authorisation and Answer Authenticity, it also ensures Test Origin Authentication. Mark Authenticity is met because the examiner posts on the bulletin board the signature of the pseudonym associated with a commitment of the mark. Notification request authentication is also met because only the candidate holds her $share_B$.

Concerning privacy, Question Indistinguishability is met by the assumptions on the origin of tests, in which the examiner generates the tests. WATA IV guarantees Anonymous Marking because the examiner cannot associate a test with a candidate until notification. Anonymous Marking can last forever, even if examiner and the invigilator collude, provided that the candidate chooses not to get her mark. Mark Privacy is met because the examiner notifies the mark to the candidate only if the latter sends a valid $share_B$. Thus each candidate can

get only their corresponding mark. Notably, the anonymiser cannot associate a candidate with a mark, because the examiner only publishes the commitment of the mark on the bulletin board. It follows that also Mark Anonymity is met.

Concerning the individual verifiability requirements, Mark Integrity is met because the candidates can verify that the examiner registered the same mark whose commitment was published on the bulletin board. Similarly, Mark Notification Integrity is met because the candidate can verify that the mark notified by the examiner is the same committed on the bulletin board.

Dispute resolution

An interesting feature of WATA IV is the support for dispute resolution during testing. In fact, the combination of signatures and visual cryptography guarantees an easy procedure to find the culprit if the candidate or the examiner misbehave. Therefore, Dispute Resolution is an accountability requirement as it allow a judge to blame the principal who misbehave in the execution of the protocol.

In WATA IV the judge is the invigilator, and the dispute originates if no intelligible pseudonym can be read when the candidate overlaps the paper sheet with the transparency sheet. Should such a dispute arise, the invigilator could then quickly resolve it as follows. First, he scans the QR code printed on the candidate's paper sheet and checks the correctness of the signature. Then, he checks if the candidate's details revealed by the signature match the ones written on the candidate's paper sheet. If so, the invigilator overlaps the visual crypto image revealed by the signature with the transparency sheet provided by the examiner. If this reveals no intelligible pseudonym then the examiner misprinted the corresponding transparency sheet. Otherwise the candidate misprinted hers. The outcome of the dispute can be double checked by repeating the procedure with the QR code printed on the transparency sheet of the examiner.

Comparison with WATA III

WATA IV brings along significant improvements compared to WATA III. In particular:

- WATA IV meets the same security requirements of WATA III augmented with Anonymous Marking and the two individual verifiability requirements of Mark Integrity and Mark Notification Integrity.
- WATA IV meets the security requirements despite a more realistic threat model as it drastically limits the tasks assigned to honest-but-curious roles.
- WATA IV can support both computer-based and computer-assisted exams, while WATA III supports only computer-assisted exams with traditional testing. This is possible because test and pseudonym are generated independently in WATA IV. At testing, the candidate chooses one of the computer devices provided at exam venue, and enters the pseudonym retrieved from the paper sheets. Of course, the use of computers at testing raises similar security of Internet-based exam, as discussed in chapter 5 for Remark!.

- In WATA IV the candidate receives part of the pseudonym by mail rather than at the exam venue.
- In WATA IV the examiner cannot register a different mark to the candidate after he learns the corresponding test, because he commits the mark at marking.
- In WATA IV any dispute between the candidate and the examiner can be solved with no efforts at testing: if an intelligible pseudonym appears, the candidate received the correct transparency sheet. If not, the signatures on the sheets reveal who misbehaved.

In the next section we propose an enhancement that removes the need of the honest-but-curios anonymiser in the design of the protocol. Moreover, we corroborate the results of the informal security analysis outlined above, with the automated verification of the enhanced protocol in ProVerif.

6.4 Removing the Need of Trusted Parties

The major limitation of WATA IV is that it requires an honest-but-curious anonymiser. Although its lightweight participation, relying on a trusted third party in the design of a security protocol introduces obvious risks. The risks can be mitigated by distributing the trust across several parties, as we did for Remark! in chapter 5, but it still requires at least one party to be trustworthy. In the domain of exams this is critical because parties typically have conflicting interests, and it may be hard to find an entity who can play the role of a TTP, as recent exam scandals confirm.

In this section, we propose a new protocol that guarantees several security properties without the need of a TTP. The protocol combines oblivious transfer and visual cryptography to allow candidate and examiner to jointly generate a pseudonym that anonymises the candidates test without the need of the anonymiser. The pseudonym is revealed only to the candidate at testing. We analyse the protocol formally in ProVerif and prove that it satisfies the same security requirements stated for WATA IV.

We minimise the roles used in this protocol to candidate and examiner. The latter also runs the tasks associated to the roles of anonymiser and invigilator in WATA IV. As we consider a corrupted examiner who can deviate from any of its assigned tasks, it follows that any of its sub-roles can be corrupted as well.

Oblivious transfer

This protocol uses oblivious transfer to avoid the participation of the honest-but-curious anonymiser. Oblivious transfer schemes allow a chooser to pick some pieces of information from a set that a sender offers him, in such a way that (a) the sender does not learn which pieces of information the choosers picks, and (b) the chooser learns no more than the pieces of information he picks. Our enhanced protocol adopts Tzeng's oblivious transfer scheme [Tze04]. In Tzeng's scheme, the chooser commits to some elements from a set, and sends the commitments to the sender. This, in turn, obfuscates all the set's elements, and the chooser will be able to de-obfuscate only the elements he has committed

to. Tzeng's scheme guarantees unconditional security for the receiver's choice, and it is efficient since it works with the sender and receiver's exchanging only two messages.

6.4.1 Description

In this protocol, we mainly revise the preparation phase, while the design of the other phases are similar to the design of WATA IV. Candidate and examiner jointly generate the pseudonym *pid* as a pair of visual cryptography shares, by means of an oblivious transfer scheme rather than via the anonymiser. Notably, also the procedure for dispute resolution requires some modification since it cannot rely on the anonymiser's signatures.

We describe the protocol in reference to the four exam phases. We assume that all remote communications are via a secure channel. Figure 6.7 illustrates preparation in the Alice-Bob notation; Figure 6.8 describes testing; Figure 6.9 describes the protocol's steps that concern both marking and notification.

In the description we assume the following public parameters:

n	length of the candidate's pseudonym
$\Sigma = \{s_1, \ldots, s_k\}$	alphabet of pseudonym's characters
$c_j \in \{0,1\}^{t \times u},\ j = 1, \ldots, k$	$(t \times u)$-pixel representation of a character
idC	candidate ID
ex	exam code
SPK_E	signing key of the examiner
M	set of possible marks
$g, h \in_{\mathcal{R}} \mathbb{G}_q$	generators for bit-commitments

Preparation

The goal of preparation is to generate a candidate's pseudonym, which is a string of n characters taken from the alphabet Σ, and to encode it into two visual cryptographic shares. Both candidate and examiner cannot know the pseudonym until they meet at testing, when the candidate learns her pseudonym by overlapping the examiner's share with hers. The underlying idea is that the candidate provides a commitment to an index into an array. The examiner fills the array with a secret permutation of the characters, and only when the two secrets are brought together is the selection of a character determined.

Part of this phase is inspired by one of the schemes used to print a secret, proposed by Essex *et al.* [ECHA09]. We tailor the scheme in such a way to be able to generate a pseudonym and to support the dispute resolution algorithm. More precisely, the main technical differences between preparation and their scheme are: (a) a modified oblivious transfer protocol that copes with several secret messages in only one protocol run; (b) the generation of signatures that will be used for accountability in the resolution of disputes.

In the following we refer to the steps outlined in Figure 6.7, 6.8, and 6.9. The protocol begins with the candidate providing a sequence of l commitments y_i to an index into an array of length k. (steps 1-2). More precisely, the parameter l, is chosen so that the $l - n$ elements can be later used for a cut-and-choose audit. The examiner can challenge the candidate to check whether the committed choices are in fact in the interval $[1, k]$. Otherwise, the examiner generates a

1. C calculates $y_i = g^{x_i} h^{\gamma_i}$ where:
   - $x_i \in_{\mathcal{R}} \mathbb{Z}_q^*$.
   - $\gamma_i \in_{\mathcal{R}} [1, k]$.
   - $i = 1, 2, \ldots, l$ with $l > n$.
2. $C \rightarrow E$: $y_1, y_2, \ldots, y_l$.
3. E calculates $\beta_{ij} \leftarrow_{\pi_{\mathcal{R}}} (\alpha_i \oplus c_j)$, $\omega_{ij} = \langle a_{ij}, b_{ij} \rangle \leftarrow \langle g^{r_{ij}}, \beta_{ij} \left(\frac{y_i}{h^j}\right)^{r_{ij}} \rangle$, $com = h^s \prod_{i=1}^{l} {g_i}^{\alpha_i}$, and $sign1 = Sign_{SSK_E}(idC, ex, com)$ where:
   - $\alpha_i \in_{\mathcal{R}} [0, 1]^{t \times u}$.
   - $s, r_{ij} \in_{\mathcal{R}} \mathbb{Z}_q^*$.
   - $g_i \in_{\mathcal{R}} \mathbb{G}_q$.
   - $i = 1, 2, \ldots, l$.
   - $j = 1, 2, \ldots, k$.

   or runs the challenge procedure against $y_1, y_2, \ldots, y_l$.
4. $E \rightarrow C$: $(\omega_{11}, \ldots, \omega_{1k}), \ldots (\omega_{l1}, \ldots, \omega_{lk})$ and $sign1$.
5. C calculates $\chi_i \in [1, l]$ and $\sigma_j \in [1, l]$ where:
   - $i = 1, 2, \ldots, m$.
6. $C \rightarrow E$: $\chi_1, \chi_2, \ldots, \chi_m$ and $\sigma_1, \sigma_2, \ldots, \sigma_n$.
7. E calculates $ev_{\chi_i} = \langle \alpha_{\chi_i}, (\beta_{\chi_i 1}, \beta_{\chi_i 2}, \ldots, \beta_{\chi_i k}), (r_{\chi_i 1}, r_{\chi_i 2}, \ldots, r_{\chi_i k}) \rangle$ and $sign2 = Sign_{SSK_E}(idC, ex, (\sigma_1, \sigma_2, \ldots, \sigma_n))$ where
   - $i = 1, 2, \ldots, m$.
   - $j = 1, 2, \ldots, k$.

   and prints $\texttt{transp} = \langle (\alpha_{\sigma_1}, \alpha_{\sigma_2}, \ldots, \alpha_{\sigma_n}), idC, ex, QR3 \rangle$ where
   - $QR3 = idC, ex, (\alpha_1, \alpha_2, \ldots, \alpha_l, s)$.
8. $E \rightarrow C$: $ev_{\chi_1}, ev_{\chi_2}, \ldots, ev_{\chi_m}$ and $sign2$.
9. C checks ev_{χ_i}, calculates $\beta_{\sigma_j} = \frac{b_{\sigma_j \gamma_j}}{(a_{\sigma_j \gamma_j})^{x_{\sigma_j}}}$ where
   - $i = 1, 2, \ldots, m$.
   - $j = 1, 2, \ldots, n$.

   and prints $\texttt{paper} = \langle (\beta_{\sigma_1}, \beta_{\sigma_2}, \ldots, \beta_{\sigma_n}), idC, ex, QR1, QR2 \rangle$ where
   - $QR1 = idC, ex, sign1$.
   - $QR2 = idC, ex, sign2$.

Figure 6.7: Preparation phase

10. $C \xrightarrow{hands} E$: `id_doc`
11. E checks `id_doc`
12. $E \xrightarrow{hands} C$: `transp`, `test`
13. C calculates $pid = (\alpha_1, \alpha_2, \ldots, \alpha_n) \oplus (\beta_1, \beta_2, \ldots, \beta_n)$ and writes $\texttt{test}_{\texttt{filled}} = (answers, pid)$
    or runs the Dispute Resolution algorithm if no pseudonym appears.
14. $C \xrightarrow{hands} E$: $\texttt{test}_{\texttt{filled}}$

Figure 6.8: Testing phase

15. E calculates $c = g^v h^{mark}$ and $sign3 = Sign_{SSK_E}(pid, c)$ where:
    - $v \in_{\mathcal{R}} \mathbb{Z}_q^*$.
    - $mark \in M$.
16. $E \rightarrow \mathcal{BB}$: $sign3$
17. $C \rightarrow E$:$(\beta_1, \beta_2, \ldots, \beta_n), sign1, sign2, sign3$
18. E calculates $sign4 = Sign_{SSK_E}(idC, ex, pid, mark, v)$
19. $E \rightarrow C$: $sign4$

Figure 6.9: Marking and Notification phases

sequence of randomly chosen $t \times u$ images, indicated as $\alpha_1, \ldots, \alpha_l$ in Figure 6.7. A sequence of k images, $(\beta_{i1}, \ldots, \beta_{ik})$, are generated from α_i and each possible character c_j. The sequence is randomly permuted and repeated for all i, resulting in l sequences of $(\beta_{11}, \ldots, \beta_{1k})$, $\ldots$, $(\beta_{l1}, \ldots, \beta_{lk})$. The secret permutation and the commitment allow that the selection of character is determined only when the two secrets are brought together.

The examiner then generates the obfuscation ω_{ij} from each β_{ij} and a commitment on each α_i, indicated as *com* (step 3), which is signed and sent with the sequences of obfuscations $(\omega_{11}, \ldots, \omega_{1k})$,$\ldots$, $(\omega_{l1}, \ldots, \omega_{lk})$ to the candidate (step 4). The obfuscation allows the candidate to retrieve only the elements whose indexes correspond to the choices she committed in step 1 (y_i).

The candidate performs a cut-and-choose audit, selecting a random set of $l - n$ sequences amongst the ω. In so doing, she can check whether the examiner generated the sequence of images correctly. The remaining substitutions $\sigma_1, \sigma_2, \ldots, \sigma_n$ select the indexes of the images that make the pseudonym. Thus, the visual share of the examiner consists of the concatenated images $(\alpha_{\sigma_1}, \ldots, \alpha_{\sigma_n})$ (step 5-6).

The examiner then generates the proofs for the cut-and-choose audit, and prints the visual share and the candidate's details in the transparency printout `transp`. This also include all the elements $\alpha_1, \ldots, \alpha_l$ and the value used for their commitment (step 7), which are stored in the form of QR code. The examiner then sends the proofs and the signed substitutions σ to the candidate (step 8). In turn, the candidate checks the proofs, de-obfuscates the elements ω, and retrieves the visual share consisting of the concatenated image $(\beta_{\sigma_1}, \beta_{\sigma_2}, \ldots, \beta_{\sigma_n})$. She finally prints the share, together with the two signatures, on a `paper` printout (step 9). At this point, both candidate and examiner have a visual share, which once overlapped reveal an intelligible sequence of characters that serves as pseudonym.

The candidate's paper printout includes two QR codes (*QR1*, and *QR2*) while the examiner's transparency only one (*QR3*). All the three QR codes share the same candidate identity *idC* and exam identifier *ex*. The QR codes *QR1* and *QR2* encode the two signatures of the examiner, respectively on commitment of the elements α and on the substitutions σ, while *QR3* encodes the elements α.

Testing

The candidate brings the paper printout at exam venue, while the examiner brings the transparencies. The examiner authenticates the candidate by checking her identity document (step 10-11). He then gives the candidate her corresponding transparency and the test (step 12). The candidate overlaps her paper printout with the transparency and learns her pseudonym, which she writes on the test (step 13). If no pseudonym appears, then this may happen only if the candidate or the examiner misprinted their printouts, and the Dispute Resolution outlined in Algorithm 1 reveals the party that is accountable for the misbehaviour. At the end of the phase, the candidate returns the filled test anywhere in the pile of tests (step 14), and takes both transparency and paper printouts home.

Marking and Notification

At marking, the examiner evaluates the answers and generates a commitment on the assigned mark (step 15). Then, he signs both mark and pseudonym found in the answer sheet, and publishes the signature on the bulletin board (step 16).

Notification opens for a fixed time, during which the candidate can remotely request to learn and register her mark. She has to send the ordered sequences of $\beta_1, \ldots, \beta_n$ and all the signatures so far she collected to the examiner (step 17). The examiner checks the signatures, overlaps the given sequence with the corresponding sequences of $\alpha_1, \ldots, \alpha_n$, and learns the pseudonym. Again, if no registered pseudonym appears, Dispute Resolution can reveal the party who misbehaved. The examiner signs the mark and the secret parameter used to commit the mark (step 18), and sends the signature to the candidate (step 19). In so doing, the candidate can verify the correctness of the mark by looking at the bulletin board.

Data: Public parameters: $(C, n, g_i, h, idC, SPK_E)$

- $paper = ((\beta_{\sigma_1}, \beta_{\sigma_2}, \ldots, \beta_{\sigma_n}), idC', ex', sign1, sign2)$ where:
  - $sign1 = Sign_{SSK_E}(idC'', ex'', com)$
  - $sign2 = Sign_{SSK_E}(idC''', ex''', (\sigma'_1, \sigma'_2, \ldots, \sigma'_n))$
- $transp = (\alpha_{\sigma''_1}, \alpha_{\sigma''_2}, \ldots, \alpha_{\sigma''_n}), idC', ex', (\alpha'_1, \alpha'_2, \ldots, \alpha'_l, s)$.

Result: Corrupted principal
if *sign1 is verifiable with* SPK_E ***and*** *sign2 is verifiable with* SPK_E ***and*** $idC = idC' = idC'' = idC'''$ ***and*** $ex = ex' = ex'' = ex'''$ **then**
  if $com \neq h^s \prod_{i=1}^{l} {g_i}^{\alpha'_i}$ ***or*** $pid{=}(\alpha'_{\sigma'_1}, \alpha'_{\sigma'_2}, \ldots, \alpha'_{\sigma'_n}) \oplus (\beta_{\sigma_1}, \beta_{\sigma_2}, \ldots, \beta_{\sigma_n})$ **then**
    return Examiner
  else
    return Candidate
else
  return Candidate

Algorithm 1: Dispute resolution

Dispute resolution

The Algorithm 1 provides an efficient way to resolve a dispute if the candidate retrieves no intelligible pseudonyms when she overlaps the visual shares. We assume that at exam venue it is available an electronic device with a camera, such as a smart phone or tablet, which stores the public key of the examiner. The input of the Algorithm are the two QR codes printed on the paper printout (QR1 and QR2) and the QR code printed on the transparency (QR3), all scanned with the camera of the electronic device.

First, the algorithm checks the correctness of the signatures encoded in QR1 and QR2. It also checks whether the candidate identity and the exam identifier

Primitive	Equation
Probabilistic symmetric enc.	$sdec(senc(m,k,r),k) = m$
Signature	$getmess(sign(m,ssk)) = m$ $checksign(sign(m,ssk),spk(ssk)) = m$
Visual cryptography	$overlap(share, gen\_share(m,share)) = m$ $overlap(share,share) = share$
Oblivious transfer	$deobf(obf(r,m,\texttt{sel},commit(r',\texttt{sel})),r') = m$

Table 6.1: Equational theory to model the enhanced protocol

reported on the paper printout match the ones in QR1 and QR2. If any one of the checks fails then the candidate misprinted her paper printout thus she is the culprit. Otherwise, the algorithm uses the data in QR3 to check the correctness of the examiner's commitment and that no pseudonym appears using the α elements indexed with the σ substitutions encoded in QR3. If any one of these checks fails then the examiner misprinted the transparency and thus he is guilty, otherwise the candidate is the culprit.

6.4.2 Formal Analysis

We analyse the protocol in ProVerif. In the remainder, we first presents the formal model, and then the results of the analysis of authentication, privacy, verifiability, and accountability requirements.

Model choices

We model TLS and face-to-face communications between the roles using the cryptographic primitive of probabilistic symmetric encryption rather than using ProVerif's private channels. This choice is motivated because the attacker cannot monitor communications via ProVerif's private channels, and cannot even know if any communication happens. We think this is a too strong assumption that may miss attacks. By renouncing to private channels, we achieve stronger security guarantees for the analysis of the protocol. Moreover, our choice has a triple advantage: it allows the attacker to learn when a candidate registers for the exam or is notified with a mark; it allows modelling either corrupted candidate or examiner by just sharing the private key with the attacker; it increases the chance that verification in ProVerif terminates. Also, the attacker has more discretional power because he can observe when a candidate is given the test and when she submits the answers.

We use the equational theory illustrated in Table 6.1 to model the cryptographic primitives of the protocol. The theory for probabilistic symmetric key consists of two functions *senc* and *sdec*. A message encrypted with a private key can only be decrypted using the same private key. Note that the randomness r on the encryption algorithm causes that the same message encrypted several times outputs different ciphertexts. The equational theory for signature is the same used in the other protocols considered in this dissertation.

We introduce a novel theory in ProVerif to model oblivious transfer and visual cryptography. The function *obf* allows the examiner to obfuscate the

elements $\beta_1, \ldots, \beta_i$, while the function *deobf* returns the correct element β_{sel} to the candidate, depending on the choice she committed. We also provide the theory for the Pedersen commitment scheme with the function *commit*. Finally, we model the generation of a visual cryptography share with *gen_share*, and their overlapping with the function *overlap*.

The process of the candidate is modelled in Figure 6.10; the process of the examiner is in Figure 6.11; the exam process is depicted in Figure 6.12. In addition, we also model an unbounded number of corrupted candidates who can register for the exam as in Figure 6.13. All the processes are augmented with the events that allow the verification of authentication requirements. We verify Anonymous Marking in presence of a corrupted examiner and corrupted co-candidates. The corresponding ProVerif process is depicted in Figure 6.14. We add the process *collector* that simulates the desk where candidates leave their tests (Figure 6.15). To verify Question Indistinguishability, we consider corrupted candidates, while for both Mark Privacy and Mark Anonymity we consider corrupted eligible candidates who can register for the exam but cannot participate at testing.

Data: Public parameters: (g, h, SPK_E)
- $sign3 = Sign_{SSK_E}(pid, c)$
- $idC, pid', mark, v$.

Result: Whether the candidate was notified with the mark assigned to her test.

if $pid = pid'$ ***and*** $c = g^v h^{mark}$ **then**
| **return** *true*
else
| **return** *false*

Algorithm 2: The verifiability-test for Mark Integrity I.V.

Modelling Verifiability. The individual verifiability definition of Mark Integrity can be modelled with Algorithm 2, which defines the verifiability-test *testMI*. The corresponding ProVerif model is depicted in Figure 6.16.

The verifiability-test *testMI* takes in the pseudonym *pid*, the randomness value of the commitment, and the mark notified to the candidate via a private channel. It also takes as input the notification *sign3* signed by the examiner from the bulletin board. The verifiability-test checks if the examiner's signature is correct and the disclosure of the commitment contained on the notification reveals the mark provided by the candidate. Since this requirement is interesting in presence of corrupted co-candidates and a corrupted examiner, we model the bulletin board as a ProVerif process to check soundness. In so doing, we can annotate the process of the bulletin board with the event `marked` in the location where it publishes the notification. We also annotate the process of the candidate with the event `assigned` in the location where the process receives the notification. We then use the following correspondence assertion to check soundness:

$$\texttt{OK}\langle id, pid, mark\rangle \rightsquigarrow \texttt{marked}\langle pid\rangle \cup \texttt{assigned}\langle id, pid, mark\rangle$$

```
let C (idC: host, k: key, dvk: key, answer: bitstring,
       choiceC: bitstring, spkE: pkey) =
(*Preparation*)
 out(ch, idC);
 (* commitment on choices *)
 new x: rand;
 let commitC= commit(x, choiceC) in
 out(ch, senc((commitC, idC),k));
 (* sign1 *)
 in(ch, encomega: bitstring);
 let (omega1: bitstring, omega2: bitstring, sign1: bitstring)=
     sdec(encomega,k) in
 let (=idC, =spkE, com: bitstring)=checksign(sign1, spkE) in
 (* cut-and-choose & retrieve beta *)
 in(ch, encsign2: bitstring);
 let (sign2: bitstring)=sdec(encsign2,k) in
 let (=idC,=spkE,alpha:bitstring,beta1:bitstring,beta2:bitstring)=
     checksign(sign2, spkE) in
  if ((code1=overlap(c0,alpha,beta1) &&
       code2=overlap(c0,alpha,beta2)) ||
      (code2=overlap(c0,alpha,beta1) &&
       code1=overlap(c0,alpha,beta2))) then
   if beta1= deobf(omega1,x) then
    Ctesting(dvk, k, answer,  beta1, idC,spkE,sign2,sign1)
   else if beta2= deobf(omega2,x) then
    Ctesting(dvk, k, answer,  beta2, idC,spkE,sign2,sign1).

let Ctesting(dvk: key, k:key, answer: bitstring, beta: bitstring,
             idC: host, SPKe: pkey, sign2: bitstring,
             sign1: bitstring) =
(*Testing*)
 in(ch, dvtransp: bitstring);
 let (question: bitstring, alpha': bitstring, =idC, =SPKe) =
     sdec(dvtransp, dvk) in
 let pid=overlap(c0,alpha',beta) in
 if pid=code1 || pid=code2 then (*otherwise dispute resolution*)
  event submitted(idC, SPKe , question, answer,pid);
  out(ch, senc((answer, pid, question),dvk));

(*Marking*)
  in(ch, sign3: bitstring);
  let (=pid, commitM: commitment)=checksign(sign3, SPKe) in

(*Notification*)
  (* request to learn her mark *)
  event requested(idC, pid);
  out(ch, senc( (beta, sign1, sign2, sign3, pid),k));
  in(ch, encsign4: bitstring);
  let sign4=sdec(encsign4,k) in
  let (=idC, =pid, =SPKe, mark: bitstring, v: rand) =
      checksign(sign4, SPKe) in
  if commitM=commit(v, mark) then
   event notified(idC, mark,pid).
```

Figure 6.10: The process of the candidate

```
let E (sskE:skey, question: bitstring, idX: host) =
 new perm_ch: channel;
 (
  (
(*Preparation*)
   get sslkey(=idX,k) in (* check if the host is eligible *)
   in(ch, enccommit: bitstring);
   let (commitX: commitment, =idX)=sdec(enccommit,k) in
   new alpha: bitstring;
   in(ch, (c': bitstring, c'':bitstring)); (* calculate beta *)
   let beta1=share(c',alpha) in
   let beta2=share(c'',alpha) in
   new r1: rand; new r2: rand; (* obfuscate the betas *)
   let omega1=obf(r1, beta1, s1, commitX) in
   let omega2=obf(r2, beta2, s2, commitX) in
   new s: rand;    (*commitment on alphas *)
   let com=commit(s, alpha) in
   let sign1=sign( (idX, spk(sskE), com), sskE) in
   out(ch, senc( (omega1, omega2, sign1), k));(*cut-and-choose*)
   let sign2=sign( (idX, spk(sskE), alpha, beta1, beta2), sskE) in
   out(ch, senc(sign2, k) );(* Print the transparency *)
   let dvtransp=(question, alpha, idX, spk(sskE)) in
   event registered(idX);

(*Testing*)
   (* de visu authentication *)
   (* get the key to emulate de visu scenario at testing *)
   get dvkey(=idX, dvk) in
   out(ch, senc(dvtransp,dvk)); (* hand the transparency *)
   in(ch, enctest: bitstring); (* get the filled test *)
   let (answer: bitstring, pid: bitstring, =question) =
       sdec(enctest, dvk) in
   if (pid=code1 || pid=code2) then
    event collected(idX, spk(sskE), question, answer,pid);
    event distributed(idX,question, answer,pid);
   (*otherwise dispute resolution*)

(*Marking*)
    new v: rand;  new mark: bitstring;
    let commitM= commit(v, mark) in
    let sign3= sign( (pid, commitM), sskE) in
    (* publish the commitment of the mark *)
    event marked(question,answer,mark,pid);
    out(ch, sign3);

(*Notification*)
    in(ch, encnotif: bitstring);
    let (beta: bitstring, =sign1, =sign2, =sign3, =pid)=
        sdec(encnotif, k) in
    if pid=overlap(c0,alpha,beta) then (* is the correct beta? *)
     event stored(idX, mark,pid);  (* notify the candidate *)
     out(ch, senc(sign((idX, pid, spk(sskE), mark, v), sskE), k))
  )  |
  (out(perm_ch, (code1,code2)) | out(perm_ch, (code2,code1)))
 ).
```

Figure 6.11: The process of the examiner

```
process
 !(
   new sskE: skey; let spkE = spk(sskE) in out (ch, spkE);

   (!in(ch, idhost: host); new questions:bitstring;
   E(sskE, questions, idhost))  |
   (!processA) |
   (! (new answer: bitstring; new k: key; insert sslkey(CA, k);
      new dvk:key; insert dvkey(CA, dvk);
      C(CA, k, dvk,  answer, s1, spkE))
   )
  )
```

Figure 6.12: The exam process

```
let processA =
 in(ch, (h: host, kX: key, dvkX: key));
 if h<>CA then
  insert sslkey(h,kX).
  insert dvkey(h,dvkX).
```

Figure 6.13: The process of corrupted candidate

```
process
 !(
   new sskE: skey; let spkE = spk(sskE) in out (ch,(sskE, spkE));
   new coll_key: key;

   (!processA) | (!collector(coll_key)) |
   (new idC: host; new k: key; insert sslkey(idC, k);
    (* The examiner is corrupted so the attacker knows the keys *)
    new dvk:key; insert dvkey(idC, k); out(ch, k); out(ch, dvk);
    C(idC, k, dvk,  choice[ansA,ansB], coll_key, s1, spkE)
   ) |
   (new idC: host; new k: key; insert sslkey(idC, k);
    (* The examiner is corrupted so the attacker knows the keys *)
    new dvk:key; insert dvkey(idC, k); out(ch, k); out(ch, dvk);
    C(idC, k, dvk, choice[ansB,ansA], coll_key, s2, spkE) )
  )
```

Figure 6.14: The exam process to analyse Anonymous Marking

```
let collector (coll_key: key) =
 in(ch, (encA: bitstring, idcA: bitstring));
 in(ch, (encB: bitstring, idcB: bitstring));
 let (answerA: bitstring, codeA: bitstring, =idcA) =
     sdec(encA, coll_key) in
 let (answerB: bitstring, codeB: bitstring, =idcB) =
     sdec(encB, coll_key) in
 if idcA<>idcB && codeA<>codeB then
  (* A candidate cannot submit a test twice *)
  let tA= (answerA,codeA) in
  let tB= (answerB,codeB) in
  out(ch,choice[tA,tB]);
  out(ch,choice[tB,tA]).
```

Figure 6.15: The collector process

```
let testMI(spkE: pkey,  priv_ch: channel, bbpk: pkey) =
 in(ch, signsign_bb:bitstring);
 in(priv_ch, (idX:host, pid':bitstring, mark:bitstring, v:rand));

 let (sign_bb:bitstring)=checksign(signsign_bb, bbpk) in
 let (pid: bitstring, commitM: commitment) =
     checksign(sign_bb, spkE) in

 if pid=pid' && commitM=commit(v, mark) then
  event OK
 else KO.
```

Figure 6.16: The ProVerif process of the verifiability-test *testMI*

```
let dispute(spkE: pkey, dvk: key) =
 in(ch, transppaper: bitstring);

 let ((q: bitstring, alpha: bitstring, idhost: host, =spkE,
       alpha': bitstring, s: rand),
      (beta: bitstring, idhost: host, =spkE, sign1: bitstring,
       sign2: bitstring)) = sdec(transppaper, dvk) in
 let (idX: host, ex': pkey, com: commitment) =
     checksign(sign1, spkE) in
 let (idX': host, ex'': pkey, alpha'': bitstring,
      beta1: bitstring, beta2: bitstring) =
     checksign(sign2, spkE) in

 if idX=idX' && idX=idhost && ex'=ex'' && ex'=spkE then
  if com<>commit(s, alpha') || code1=overlap(c0,alpha',beta) ||
     code2=overlap(c0,alpha',beta) || alpha'<>alpha'' then
   event Eguilty
  else event Cguilty
 else event Cguilty.
```

Figure 6.17: The ProVerif process of *dispute*

To prove completeness, ProVerif checks that the verifiability-test process does not emit the event `KO` when the input data is correct.

It can be observed that Mark Notification Integrity I.V. can be modelled with the same test used to check of Mark Integrity I.V. In fact, the candidate checks if the commitment associated to her pseudonym contains the same mark that the examiner notified to her.

Modelling Dispute Resolution. We introduce the first formalisation of accountability requirement for exams, namely *Dispute Resolution*. Accountability allows us to identify which principal is responsible for a protocol failure. In the case of exam, a candidate should be able to submit a test and receive the corresponding mark. If she fails in any of these, Dispute Resolution prescribes that the participant who caused such failure can be identified. We formally model Dispute Resolution with a similar approach advanced for individual verifiability, with two differences: first, we resort to unreachability of an event to prove soundness; second, we check that the exam process does not execute the algorithm of Dispute Resolution to prove completeness.

The process *dispute* is illustrated in Figure 6.17. The process takes in the examiner's transparency and the candidate's paper, and outputs the corrupted principal according to Algorithm 1 specified in section 6.4.1. It is annotated with the event `Cguilty`, which the process emits when the candidate is considered to be the culprit. The process is also annotated with the event `Eguilty`, which is emitted when the examiner is considered to be the culprit.

If the protocol executes the process `dispute`, then either the examiner or the candidate is corrupted. Thus, regarding soundness, the idea is to check that

`dispute` cannot return an honest principal instead of the corrupted one. This is captured by the following definitions.

Definition 43 (Soundness (corrupted examiner)) *The process* dispute *is sound with respect to a corrupted examiner and honest candidate if the event* `Cguilty` *is emitted in no execution trace of the exam protocol.*

Definition 44 (Soundness (corrupted candidate)) *The process* dispute *is sound with respect to a corrupted candidate and honest examiner if the event* `Eguilty` *is emitted in no execution trace of the exam protocol.*

To prove completeness, we check that the exam protocol never runs the process *dispute*, hence both events `Cguilty` and `Eguilty` are not emitted. This is captured by the following definition.

Definition 45 (Completeness (exam process)) *The exam protocol is complete respect to Dispute Resolution if neither the event* `Eguilty` *nor* `Cguilty` *are emitted in any execution trace of the protocol with honest roles.*

We thus can say that the exam protocol ensures Dispute Resolution if `dispute` is sound and the exam protocol complete.

Limitations

A limitation of the formal model is the specification of the cut-and-choose audit due to the powerful ProVerif's attacker model. In fact, if the attacker plays the cutter's role, he might cut the set of elements such that the subset audited by the chooser is correct, while the other subset not. Although in reality the probability of success of this attack for a large set of elements is small, it is a valid attack in ProVerif irrespective of the number of elements. In our case, the chooser is the candidate and the cutter the examiner. We thus have a false attack when the examiner is corrupted, namely controlled by the attacker. In this case, we avoid this situation by allowing the candidate to check all the elements of the set. This is sound because the candidate plays the role of the chooser, thus she is honest and follows the protocol although she knows the extra information.

Results

Table 6.2 outlines the results of our analysis. ProVerif confirms that the protocol guarantees all the authentication requirements despite allowing an unbounded number of corrupted eligible co-candidates. Thus, the exam protocol ensures authentication although the attacker can register to the exam.

Concerning privacy properties, ProVerif proves that the exam protocol guarantees all the privacy requirements. We do not consider Anonymous Examiner since we assume only one examiner, hence the protocol trivially fails to meet the requirement.

ProVerif confirms the verifiability-test *testMI* is sound and complete, thus the exam protocol is Mark Integrity I.V. and Mark Notification Integrity I.V. verifiable.

Finally, the exam protocol ensures Dispute Resolution: ProVerif shows that the protocol does not blame honest principals when the `dispute` algorithm is

Requirement	Result	Time
Candidate Authorisation	✓	8 s
Answer Authenticity	✓	7 s
Test Origin Authentication	✓	7 s
Test Authenticity	✓	8 s
Mark Authenticity	✓	8 s
Notification Request Auth.	✓	8s
Question Indistinguishability	✓	<1 s
Anonymous Marking	✓	27 s
Mark Privacy	✓	28 m 41 s
Mark Anonymity	✓	52 m 12 s
Mark Integrity I.V.	✓	<1s
Dispute Resolution	✓	<1s

Table 6.2: Summary of the analysis of the enhanced protocol

executed (soundness), and that does not run the algorithm when both examiner and candidate are honest (completeness).

6.5 Conclusion

This chapter draws its motivation by observing that exam security has a major role in the widespread acceptance of computer-assisted exams. It focuses on a family of computer-assisted exams called WATA, and shows how to gradually remove the need of trusted third party in their design, though ensuring more security requirements.

This chapter provides a new outlook of WATA II and WATA III, which originally were conceived as software running into the examiner's machine, by re-engineering them as exam protocols. It discusses their security properties, trust assumptions, and both security and functional limitations. Then, it advances WATA IV, a novel exam protocol that allows both remote registration and remote notification. WATA IV reduces significantly the participation of TTP respect to the previous versions, while guaranteeing more security requirements. Moreover, WATA IV supports both computer-based exam and traditional testing.

WATA IV is further enhanced resulting in a new protocol that meets the same security requirements of WATA IV but without the need of a TTP. The underlying idea is to combine oblivious transfer and visual cryptography to generate a pseudonym that anonymises the test for the marking. A formal analysis in ProVerif confirms that the enhanced protocol ensures all the stated requirements.

Finally, this chapter advances the accountability requirement of Dispute Resolution and its formal specification. Using ProVerif, we prove that the enhanced protocol also meets such requirement without the need of a TTP.

Chapter 7

Formal Analysis of Certificate Validation in SEB and Modern Browsers

Computer-assisted exams often include some remote tasks, such as remote registration or remote notification of candidates, which normally take place by means of a browser. Internet-based exams also require that candidates take the exam via software, possibly a browser [SBL+12] [Com15, Fid15]. Although these browsers are customised to prevent unauthorised access to resources during the exam, they still appeal on TLS to meet authentication and privacy. In the design of secure exam protocols, as well as in many security protocols, it is often assumed that an implementation of TLS channel is available to secure the communications among the principals.

While we can reasonably assume that TLS provides privacy and integrity as it uses robust cryptographic schemes, we should be more careful in assuming the same for authentication. The authentication of a website depends, to various degrees, on trust. One element of trust comes either from the web-of-trust concept [Zim92] or from the public-key infrastructure (PKI) [CSF+08]. Either way, the authentication of a website works through the emission of certificates. A certificate binds an identity with a public key, and contains other pieces of information that the verifier, also known as authenticator or trustor, needs to check to accept the certificate. The verifier is a software, normally the browser of the user who accesses a website. The browser verifies that the identity on the certificate corresponds to the identity on the website, and that the certificate is signed by a trusted authority. Thus, an authentication that succeeds seems to depend mostly on the browser than on the user. But when the validation fails, browsers usually resort on the choices of users. In this case, the user is the ultimate responsible for the website's authentication.

Invalid certificates are not rare. For example, exam authorities and institutions often self-issue their own certificates rather than purchase them from accredited certification authorities. Self-issuing a certificate is in fact cheaper. But, even if institutions purchase their certificates from a recognised authority, they may still use the certificate beyond its expiration date or abuse it to certify different domains and sub-domains that are not actually provided for the

certificate. With a security take, an invalid certificate may originate from a network attacker who attempts a man-in-the-middle attack, namely the attacker replaces the server's certificate with his own.

We do not intend to contribute to the long-established debate on the interpretation of the technical meaning of authentication [Gol96]. We rather observe that browser's security is critical for exams, and we expect to substantiate our observation that server authentication goes beyond the technical *certification path validation* algorithm as described in the standard X.509 [CSF+08]. The validation of a certificate is in fact *socio-technical* as modern browsers consider user's choices and support the validation with novel technologies, such as HSTS. Thus, with *certificate validation* we refer to the extended protocol that browsers customise with user's involvement and additional security mechanisms.

We note that the way the certificate validation is accomplished varies considerably among browsers. This variety motivates a number of research questions:

- What are the differences in terms of user involvement in how modern browsers implement server authentication?
- Which browsers reduce the security risks for users when a certificate is invalid?
- Can browsers improve their security by involving the user more profitably than they do at present?

This list of questions, purposely truncated to length three here, arises when server authentication, and certificate validation in particular, is assessed from a socio-technical standpoint.

This work complements traditional human-computer interaction studies by advancing what seems to be the first formal analysis of browser's certificate validation that is not only logically conditioned on the technology but also on user actions.

In this chapter, we consider six browsers. One is Secure Exam Browser (SEB), which specifically addresses the security of remote testing of Internet-based exams. The others are Firefox, Chrome, Safari, Internet Explorer, and Opera Mini, which are the most popular browsers and may support the remote tasks of an exam. We also consider the analysis of private browsing mode, and the interleaving of classic and private browsing for the browsers that support it. We introduce five socio-technical requirements each binding elements like TLS session, certificates and, notably, user choices. The mix of browsers, modes, and requirements allows for the analysis of 60 different scenarios. We anticipate that the results of the analysis turn out to be interesting, from which we are able to state four recommendations. Notably, the results of our analysis include two bugs that concern Safari. We reported the bugs to Apple, and were replied that a fix would be available in the upcoming versions of Safari for iOS and OS X.

The contribution of this chapter exceeds the formal analysis of the 60 scenarios. This formal analysis was not carried out using a known approach. By contrast, it was not obvious how to represent (portions of) the functioning of a browser in order for the analyser to quickly get to grasps with its properties without reading long prose. Various graphical notations were tried out, and finally we found Unified Modelling Language (UML) activity diagrams [OMG11] to bear the necessary flexibility. Building these diagrams is a major hallmark

in our understanding of the technicalities of the browsers. However, they are only semi-formal and not directly executable; a formal model is needed for a fully automatic analysis. We therefore translate our UML diagram models to models in the CSP process algebra [Hoa78]. The model is then extended with an Linear Temporal Logic (LTL) specification of the requirements of interest. Each extended model forms the input to the Process Analysis Toolkit (PAT) model checker [SLDP09].

Outline of the chapter. Section 7.1 discusses the related work about formal approaches for the security analysis of browsers and certificate validation, and related studies of human aspects. Section 7.2 details the different aspects of certificate validation. Section 7.3 describes the SEB kiosk exam browser. Section 7.4 describes the five most popular general-purpose browsers. Section 7.5 details the browsers' certificate validation ceremonies using UML Activity Diagrams. Section 7.6 concerns the analysis of the five socio-technical requirements on the browser's ceremonies with the PAT model checker. Section 7.7 discusses the results of the analysis. Section 7.8 explores some implications and concludes the chapter.

7.1 Related Work

A few works have developed formal verification techniques to model and analyse web browsers. Akhawe *et al.* [ABL+10] introduce a formal model of web security. They define the main components of the web, namely Non-Linear Time, Browser, Servers, and the Network as *Web Concepts*. They consider a spectrum of threats that span from a malicious web server to a more advanced attacker who is able to inject contents into an honest web server. Finally, they analyse two security requirements, i.e., security invariants and session integrity, in five web security *mechanisms*. The considered mechanisms include neither TLS nor certificate validation, and the formal model assumes the user correctly interprets the browser's security indicators. Our work focuses on certificate validation and considers users who may not correctly understand security indicators. Groß *et al.* [GPS05] propose a formal framework to model a web browser and the behaviour of a user who interacts with the browser. They validate their framework by analysing the security of password-based user authentication. Contrary to modelling an ideal browser, we analyse the actual implementation of modern browsers. It would be interesting to merge our approach with their model of web browser. Formal verification techniques have been used to analyse Human-Automation Interaction [BBS13]. They mostly focus on cognitive aspects of users rather than on the technical systems they interact with. We consider user choices, which do depend on human cognitive factors, but our focus is more on the technical part, namely on the web browsers.

There are many studies that conduct a security analysis of certificate validation. Georgiev *et al.* [GIJ+12] analyse certificate validation in the context of TLS for different web applications, such as shopping carts, cloud storage, and payment gateways. They find that several APIs perform certificate validations that do not follow any standard. The authors show how such customisation leads to security vulnerabilities. Differently to our automated approach, they detect attacks by visual inspection of the source code; moreover, the analysed

applications involve no browsers. Kaminsky *et al.* [KPS10] discuss different attacks against the certificate infrastructure. They point out that certificate issuers and browsers may differently interpret the subject name in an X.509 certificate. Such a lack of standardisation makes the subject name vulnerable to injection attacks. A recent update to the X.509 standard [Yee13] aims to fix the issue. Clark and Van Oorschot [CvO13] provide a comparative evaluation of enhancements implemented into browsers for certificate validation. They argue that it is becoming more common that attackers own valid certificates for a website. Attackers' attention focuses on certificate infrastructure because of its reliance on human factors. We share their view that certificate validation goes beyond the mere binding between a domain name and a public key.

Different solutions have been proposed to modify the structural flaws of the certificate infrastructure [Mar15, MP13] [HS12], but they also require dedicated modifications to the protocols that use certificates to achieve authentication. For example, the Certificate Transparency project [LLK13] suggests an improvement to the current TLS certificate system by providing supplemental monitoring and auditing services via certificate logs. The technique requires a different server implementation to accommodate a TLS extension. Structural solutions such as TLS extensions may take a long time before being implemented extensively, because both browser and server need to support the extension. We also end up with a similar proposition, but it requires a modification of the browser only, with no change required on the server and on the TLS protocol.

The human aspect of certificate validation has recently been analysed via different empirical studies. Akhawe and Porter Felt [AF13] make an empiric analysis to assess the effectiveness of browser security warnings. In particular, they measure the users' click-through rates on certificate and malware warnings. They find that users tend to ignore warnings as they click through the Chrome certificate warnings. Such finding led Google to redesign Chrome's certificate warnings. Similarly, Flinn and Lumsden [FL05] conduct a survey to assess whether users are aware of the security risk they face online. They find that users have different interpretations of the term "secure website" and are generally unaware that TLS provides server authentication. By contrast, our formal analysis does not pertain to user perception, but investigates the various ways in which user's choices influence server authentication. Jøsang *et al.* [JVRK12] point out that web browsers can only do syntactic server authentication, as TLS cannot provide semantic server authentication. Thus, the attacker can exploit semantic attacks to trick the user. They advocate the need of a framework to determine the assurance level of server authentication. Our approach aims at analysing how web browsers help users to avoid such server authentication attacks. Gajek *et al.* [GMSS08] propose a new authentication protocol that consists of a mix between the Password Authentication Key Exchange (PAKE) and TLS protocols without relying on a PKI. They formalise a user as a probabilistic machine. The user's behaviour can recognise the so-called human-perceptible indicators like pictures and sounds. In contrast, we are not interested in the cognitive aspects, and make minimal assumptions about the user capabilities. Akhawe *et al.* [AAVS13] produce a taxonomy of certificate validation warnings and collect data over more 10 billion TLS connections that are not under MITM attacks. Thus, they calculate the false positive rate of showing warnings and present a number of recommendations to improve browser design decisions. Conversely, our approach considers MITM attacks.

The security of browsers has been studied variously, for example to avoid the user's oversight of warning messages [SEA+09], or to improve the readability of their contents [BvOP+09]. These works are positioned over the cognitive aspects of human-computer interaction with the browsers. To position our work, it is useful to note that we see the socio-technical system consisting of a web server, a computer network, a browser, a user and possibly an intruder as a *ceremony* in the sense of Ellison [Ell07]. The various technical and social layers of a ceremony have been recently identified [BCK12], with a practically useful suggestion that certain layers can be neglected, namely virtually compressed, during the analysis to sharpen the analyser's focus on other layers.

7.2 Basics

This section clarifies a few technical notions and sets the terminology used throughout the chapter. First it details the constituents of a web certificate and explains how their validation works or can fail. Then, it discusses the regulations that specifies the *path validation* of a web certificate, and observes how the standard eventually leaves the interpretation of certificate validation to browser manufacturers. In consequence, certificate validation becomes a mix of user's choices and technical security mechanisms, from which it emerges that certificate validation in modern browsers is a socio-technical process. The sequel of this Section outlines a few basic concepts needed later.

Web certificates

A web certificate binds an identity to a public key. The X.509 standard [CSF+08] specifies the structure of a certificate by listing a set of mandatory and optional fields. Among the mandatory fields, four are fundamental to understand the authentication purpose of a certificate: *subject*, which specifies the certificate owner's identity; *subject public key*, which specifies the public key associated to the subject; *issuer*, which specifies the entity who verified that the public key belongs to the owner described in the subject; *certificate signature*, which specifies the digital signature generated by the issuer on subject and public key.

Other typical mandatory fields are the following: *version*, which specifies whether optional fields are expected to be used; *serial number*, which is unique among the certificates generated by the issuer; and *validity*, which specifies the time interval during which the issuer maintains information about the status of the certificate.

Certificate validation

The algorithm for the path validation of a certificate checks whether the certificate is valid. It consists in verifying that the signature is correct provided that the verifier trusts the issuer. In this case the public key can be used to communicate with its owner. However, even if the signature is correct, the verifier may not trust the issuer. In this case the issuer, which is known as *intermediate authority*, needs itself to be certified. This forms a chain of intermediate authorities called *certification path*. The certification path always chains up to a root called *certification authority (CA)*, whose certificate is self-signed, namely

the issuer and the subject coincide. The verifier is assumed to trust the public key of the CA. Thus, the validation of a certificate path is to check the fields of each certificate up to a trustful root certificate. The standard X.509 details a certification path validation algorithm, but verifiers are free to implement their own algorithms, provided they offer equivalent functionality [CSF+08].

Invalid certificates

If the path validation of the certificate succeeds then the certificate is valid, namely the certificate binds correctly the identity with the public key. However, the validation may fail due to a number of errors:

Unknown or untrusted certificate issuer. The certificate path chains up to a certification authority that is not in the list of CAs the verifier trusts. Verifiers may trust different certification authorities, since there is no universally trusted list of CAs.

Possible reasons. A certificate may be invalid because entities, such as web servers, may prefer to self issue a certificate rather than purchase expensive certificates by commercial CAs. This choice may be even necessary when a single entity owns many different domains that need to be certified. Self-issuing a certificate is a quick procedure and has no costs. Self-issued certificates are widespread through a large number of public institutions, such as universities, or the US Army [U.S15].

Expired certificate. A certificate expires after a specific time interval. Specifically this happens because the *validity* field contains a past date.

Possible reasons. Entities may forget to renew their certificates before they expire. According to a recent survey [AAVS13], expired certificates are the most common form of benign (i.e., false positives) certification path validation failures.

Revoked certificate. A certification authority may revoke the certificate due to either administrative or security reasons. For example, if an entity believes that an attacker has learned the private key, it may ask the CA to revoke the certificate.

Possible reasons. Again, verifiers may trust different certificate authorities, thus revocation also depends on the specific CA store used by the verifier. Moreover, certificate revocation is a protocol by itself: CRL, OCSP, and CRLSets are three common protocols to revoke and check certificates [Yee13, SMA+13] [Goo12], and verifiers may use any of these.

Mismatched certificate subject. The *subject* expected by the verifier mismatches the one shown in the certificate. According to a large-scale survey on certificates [VFBH13], mismatched certificate subject is a frequent case of certificate path validation failure.

Possible reasons. False positives may occur because an entity needs to secure its own sub-domain (e.g., `www.sub1.entity.com`), but, to save costs, the entity does not purchase a certificate for each sub-domain.

The X.509 standard says that if any one of the checks of the certification path validation fail, the algorithm terminates, returning a failure indication to the concerned protocol. The TLS protocol uses X.509 certificates to support authentication, and browsers implement TLS over HTTP to provide confidentiality, integrity, and authentication on the communications with web servers. Since authentication is an optional TLS requirement, the corresponding RFC standard [DR08] outsources the certificate validation to the browsers: *"How to interpret the authentication certificates exchanged is left to the judgement of the designers and implementors of protocols that run on top of TLS"*. Moreover, the HTTP over TLS standard [Res00] advocates the involvement of the user when the certification path validation fails: *"User oriented clients MUST either notify the user (clients MAY give the user the opportunity to continue with the connection in any case) or terminate the connection."*. Ultimately, browsers can implement differently the certificate validation, which becomes socio-technical when the technical approach, namely the certification path validation, fails.

Socio-Technical aspects of certificate validation

Browsers communicate with the user through different ways, such as text warnings, pop-up windows, open or closed padlocks, and coloured address bars. The main component that browsers use to interact with the user is the viewport, which is depicted in Figure 7.1

The user can choose any of the options proposed in the browser's viewport: a cautious user may close the browsing session, while a curious one may click through a warning. Users who interact with a browser may be variously skilled and educated. They are influenced by a huge variety of local or global cultural values. Malicious websites nowadays can use scripts to gain major control on browser's viewport and deceive user's [CvO13] on security warnings. Although the design of browser's security indicators has improved over the years [YSA05], a number of studies have shown that users tend to click through a warning without paying attention [Lan12] [AAVS13, SEA+09]. These social factors make the problem of certificate validation harder than it seems, namely a security problem that cannot be solved by purely technical means.

A few proposals have been recently advanced to minimise the participation of users in certificate validation and make security less reliant on user's choice [The11] [HJB12]. Different browsers have adopted HSTS [HJB12], a security mechanism originally conceived to thwart TLS stripping attacks [Mar09b, Mar09a]. In a TLS stripping attack, the attacker forces the user to communicate via an HTTP connection although the web server supports HTTPS connections. HSTS-compliant browsers prevent "unsecured" HTTP connections to HSTS-compliant web servers. To do so, the web server sends the browser an HSTS header during a secured TLS session. Optionally, the HSTS header may include a list of the only certification authorities allowed to issue the web server's certificate. Then, the browser adds an internal policy stating that the concerned web server must be accessed via HTTPS only, and with a valid certificate. Thus, once the authentication of the web server succeeded, it shall not fail in the future.

HSTS avoids user's participation in favour of a purely technical enforcement of security: if the certificate validation fails for a browser-known HSTS-complaint web servers, the browser shows the user an error message (not a

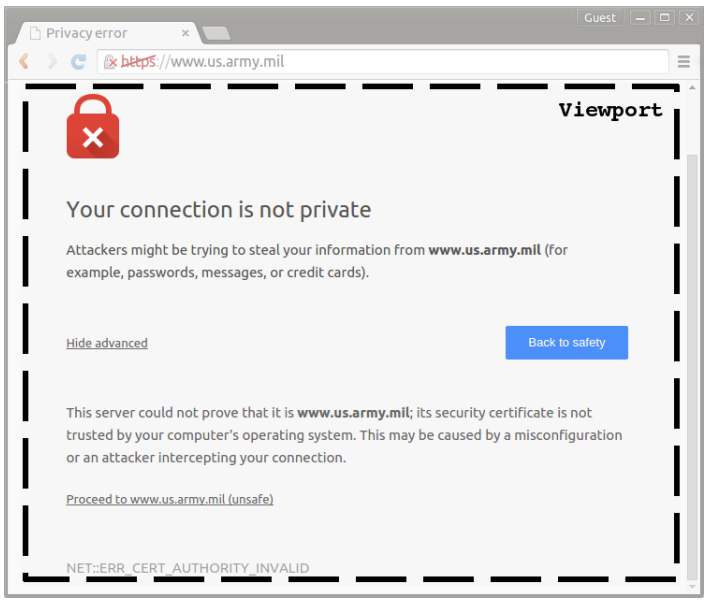

Figure 7.1: The viewport component of a browser

warning) and aborts the connection. Some browsers implement a pre-loaded whitelist of HSTS-complaint web servers to mitigate bootstrap attacks. However, similar to the certification path validation, browsers implement HSTS differently from each other, as we shall see later.

7.3 Safe Exam Browser

Safe Exam Browser (SEB) is a kiosk browser developed at ETH Zurich [SBL+12]. Exam authorities can employ SEB at testing to thwart candidate cheating in computer-based and Internet-based exams. The main functionalities of SEB allow the candidate (a) to connect to a remote web-based exam system and (b) to access to a selection of third party applications and web sites during testing. SEB prevents the candidate from accessing resources and utilities of the operating system, switching to undesired applications, quitting the browser any time, and opening a website not allowed by the authority.

SEB turns the host computer device in a kiosk browser for exams by removing any other components of a browser but the viewport. It can be installed in a computer device either provided by the exam authority or personally owned by the candidate. In the latter scenario, the browser is secured by a signed and encrypted exam configuration. Moreover, SEB supports remote testing by means of a *SEB Server* that provides more security on unmanaged computer devices thanks to screen recording and logging of all kinds of activities on the host computer. However, we stress that the threat model considered in this chapter does not concern malicious candidates or users. Rather, we focus on the security risks that a malicious observer can pose to the human principal via the browser. Similarly, screen recording and logs do cause concern about privacy, but this is a requirement not considered in this chapter.

Over the 10% of all written exams at ETH Zurich have been carried out using SEB in 2014. Besides, SEB has already been used for remote testing [SBL+12]. The particular purpose of this browser of minimising the participation of the user makes interesting to check how this can affect certificate validation.

Since SEB is open source, we studied it by looking at its official documentation and source code. We consider the version 2.0 of the browser.

7.4 Modern Browsers

We also consider the most popular browsers available nowadays. According to StatCounter [Sta15], the most used browsers are Firefox, Chrome, Internet Explorer, and Safari. Opera Mini is the most popular platform-independent browser [Ope14] and is available for many mobile devices. Its analysis is motivated because nowadays more and more users prefer to browse the Web with touchscreen mobile devices, and in particular Opera Mini dramatically reduces the amount of data transferred. In doing so, we aim to evaluate how such restrictions affect certificate validation.

Firefox. The inception of Mozilla Firefox originates from Netscape Navigator. According to StatCounter [Sta15], it is the third most popular browser over desktop, mobile, tablet, and console devices. Among the browsers we consider, Firefox seems to be the most complete: it supports HSTS, distinguishes two different certificate stores, and allows users to store server certificates either permanently or temporarily. Since Firefox is open source, we studied it by looking at its official documentation and source code. We consider Firefox version 36.0.4.

Chrome. Although Google Chrome is the youngest browser we consider, it is the most popular. It was the first browser to support HSTS policies, and adopts different certificate stores depending on the operating system underlying the browser. Chrome is based on the Chromium open source code with minor differences. We analysed Chrome inspecting the Chromium source code and using empirical tests. This work considers Chrome version 41.0.

Internet Explorer. Microsoft Internet Explorer was the most popular browser for years, and has been overtaken by Chrome only recently. Currently, it does not support HSTS, which is however planned to be implemented soon [Mim14]. Internet Explorer is available only for Windows operating systems, and uses their certificate stores. Since Internet Explorer is closed source, we relied on empirical tests, also supported by network analysers. The version of Internet Explorer analysed is 11.0.16.

Safari. Safari is the browser developed by Apple and is popular on the company devices. It supports a HSTS and, similarly to Firefox, distinguishes two different certificate stores allowing users to store server certificates. Safari is available only on Apple's operating systems and is closed source. We thus analysed it empirically and assisted by network analysers. We studied Safari 8.0.3.

Opera Mini. Opera Mini is used by more than 244 million people per month and is particularly popular in emerging countries, according to the company data [Ope14]. It aims at being the most lightweight browser for any java-capable device. To do so, the browser uses Opera proxy servers and compression technologies to reduce traffic and speed up page display. Thus, although communications to the proxy server are encrypted, there is no end-to-end TLS encryption between the browser and the web server. We analysed the (closed-source) browser empirically, using a java emulator with network analysers. In this work we consider the version 7.6.4.

7.4.1 Private Browsing

All the browsers we consider in this chapter support *private browsing*[1], which is a privacy protection mode that disables browser's history and cache. Private browsing gives the user no guarantee about Internet privacy as an eavesdropper can still learn the websites visited by the user. Rather, private browsing protects user's privacy only over the data stored in the local machine. Each browser implements private browsing differently, and this is usually done by inhibiting different features [ABJB10]. Private browsing is becoming increasingly popular among users [Bur12], so we consider it in our analysis. In particular, it is interesting to study how browsers balance security (e.g., HSTS, user's approved certificate) with privacy technologies. Concerning certificate validation, one would expect no differences between private and classic browsing. However, as we shall see later, this is not true.

Technical notes. We tested the certificate validation in SEB, Firefox, Chrome, and Internet Explorer using an Intel Core i7 3.0 GHz with 8 GB RAM running Windows 8.1 on a virtual machine. We tested certificate validation on Opera Mini inside MicroEmulator, a Java implementation of JavaME, and analysed certificate validation on Safari on an Apple MacBook Pro Intel Core i5 2.5 GHz with 8 GB RAM running OS X Yosemite 10.10. The network analysers we used to understand how certificate validation works, especially on closed-source browsers, are "Wireshark" [Ger15], "mitmproxy" [Ald15], and "Charles" [Kar15]. They ran on a second virtual machine with Linux Ubuntu 14.04 and intercepted any traffic between a server and the target browser on the main virtual machine.

7.5 Modelling Certificate Validation

As seen in Section 7.2, the certificate validation is a protocol rather than an algorithm, which may involve users. Therefore, we refer to is as a ceremony.

The certificate validation ceremony includes a user, the browser, and the web server. In this section, we provide a formalisation of the ceremony for each browser. However, finding the right formalism for the socio-technical analysis is not easy. The standard notation to describe security protocols is the Alice-and-Bob notation [LSV15]. Although this notation provides a simple and clear description, it comes with some limitations: it cannot capture fork, join, and

[1] SEB supports private browsing sessions only

branching, which are essential, for example, to model multiple user's choices in the certificate validation. Flowcharts offer a graphical description and look suitable to describe browsers and algorithms in general, but are less appropriate for the description of protocols: we have the three roles of browser, user, and server, and we need to detail the messages that the roles exchange. Message sequence charts extend flowcharts to the domain of protocols, emphasising the interaction among the roles. Although message sequence charts have been extended with formal semantics [MR94], they still have the same limitation of the Alice-and-Bob notation.

Thus, we choose the semi-formal and graphical description of UML Activity Diagram [OMG11]. An UML activity diagram is a graphical scheme that defines the activities needed to meet a given functionality. UML activity diagrams are made of shapes that model choices, interactions, and concurrency. Table 7.1 recalls the main UML activity diagram's shapes.

The contribution of activity diagrams is threefold. First, they give an intuitive representation of a protocol session, highlighting the mechanisms used on each role. Second, they can represent parallel actions (fork/join) and multiple choices (branching). Third, they can be easily translated in a fully formal language, thanks to their semi-formal semantics [AST13]. In particular, we translate activity diagrams to CSP# [SLDP09], a modelling language that enriches the high-level operators of CSP (e.g., choices, interleaving, hiding, etc.) with low-level programming constructs (e.g., arrays, while, etc.). The CSP# code is then fed to an automatic tool that checks whether the input model guarantees a set of properties. The code can be found in Appendix A and in [Giu15].

7.5.1 UML Activity Diagrams for Certificate Validation

Shape	Description
	Activity node
	Object (datastore) node
	Decision or merge point
	Input and output objects of activities
	Distinguishing the activities by role
	Initial node
	Activity final node
	Activity final node within a role
	Control flow within a role
	Control flow among different roles
	Object flow from or to an object node

Table 7.1: Description of the shapes defined in UML Activity Diagram.

We build nine activity diagrams that model the certificate validation ceremonies for the browsers both in classic and private browsing. Although we consider six browsers, we do not model the classic browsing for SEB since it supports only private browsing, and we do not model the private browsing of Internet Explorer and Opera Mini because they are identical to the corresponding classic browsing modes.

The UML Activity Diagram for SEB is in Figure 7.2; the diagrams modelling Firefox in classic and private browsing are respectively in Figure 7.3 and in Figure 7.4; the diagrams modelling Chrome in classic and private browsing are respectively in Figure 7.5 and 7.6; the diagrams for Safari are in Figure 7.7 and 7.8; the diagrams for Internet Explorer and Opera Mini are respectively in Figure 7.9 and 7.10.

The UML activity diagrams include the functionalities limited to describe how each browser achieves certificate validation. Each activity diagram has four columns, each representing a communicating role. From left to right we have the *user*, the *browser user interface*, the *browser engine*, and the *server*. Each role begins with a filled dot that points to their first activity. We assume that the browser bootstraps with the start web page, which is displayed in the browser user interface. Next, the browser user interface can load a web page because the user types a URL or clicks on an active link in the currently displayed web page. A label close to a thick arrow defines the object exchanged between activities. Activities may need to get access to datastores, which are represented as object nodes.

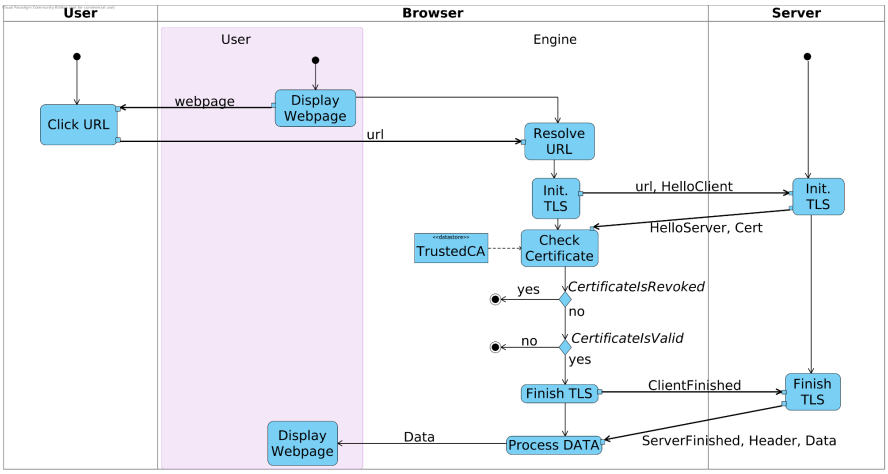

Figure 7.2: Activity diagram for certificate validation in SEB

7.5.2 Description of the Main UML Activities

For each role involved in the certificate validation, we describe the principal activities and checks that concern their UML activity diagrams. In the remainder, we denote UML activities in serif and UML decisions in *italics*.

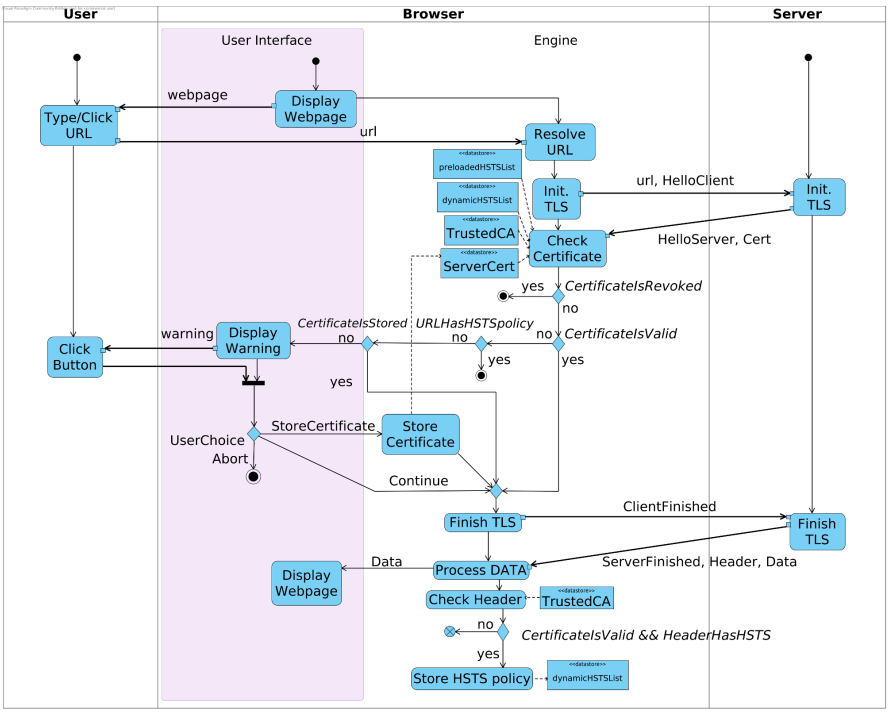

Figure 7.3: Activity diagram for certificate validation in Firefox

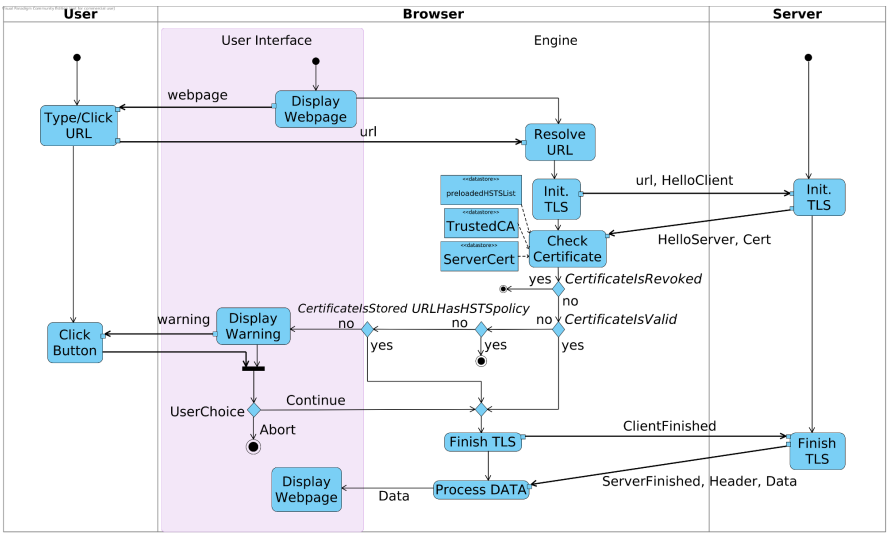

Figure 7.4: Activity diagram for certificate validation in Firefox in private browsing

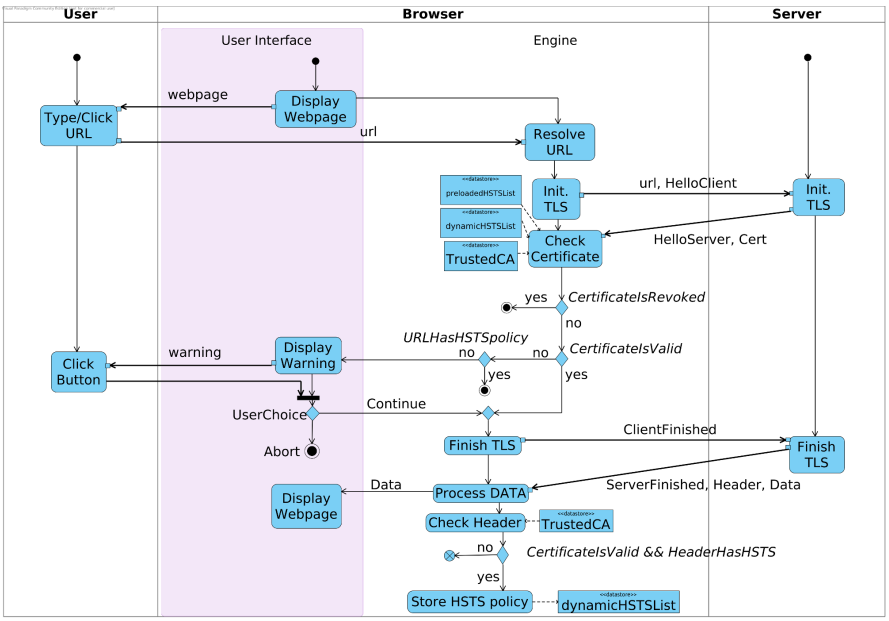

Figure 7.5: Activity diagram for certificate validation in Chrome

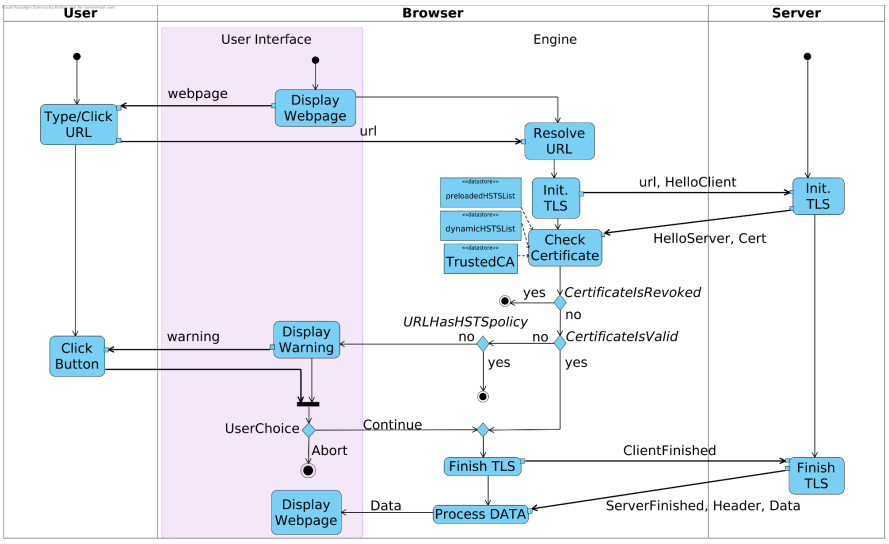

Figure 7.6: Activity diagram for certificate validation in Chrome in private browsing

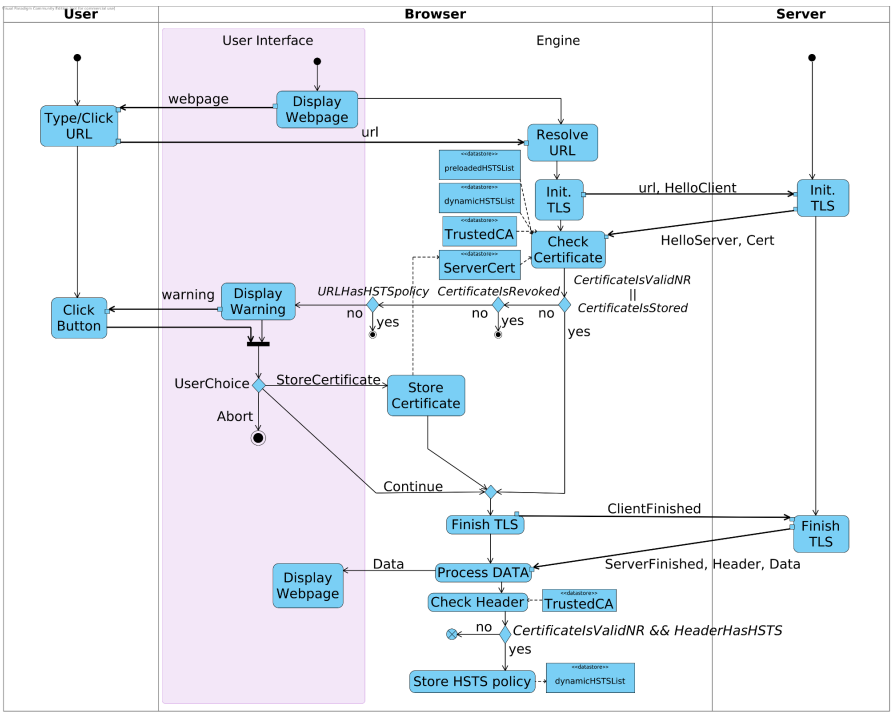

Figure 7.7: Activity diagram for certificate validation in Safari

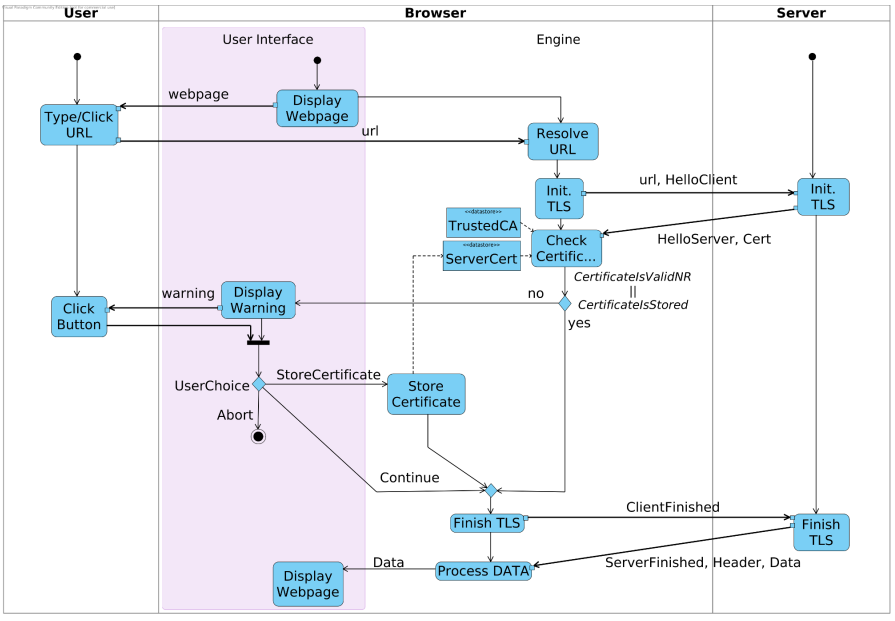

Figure 7.8: Activity diagram for certificate validation in Safari in private browsing

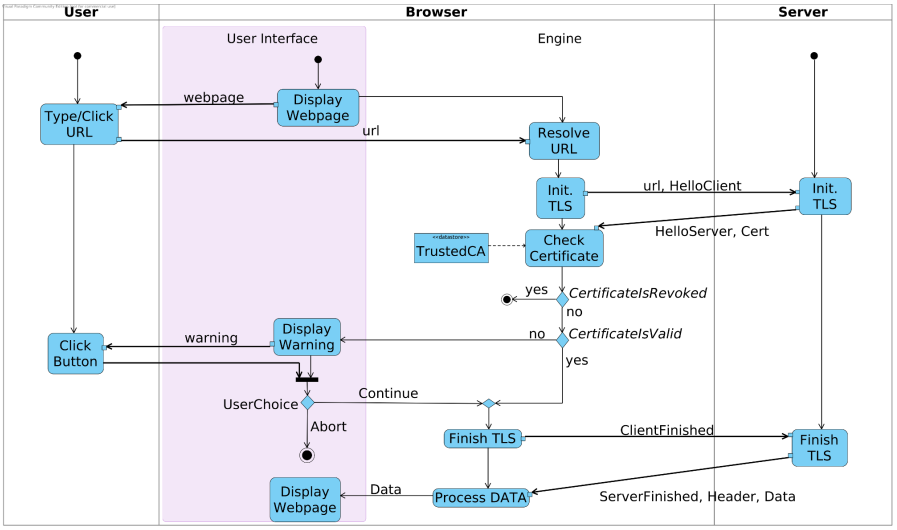

Figure 7.9: Activity diagram for certificate validation in Internet Explorer

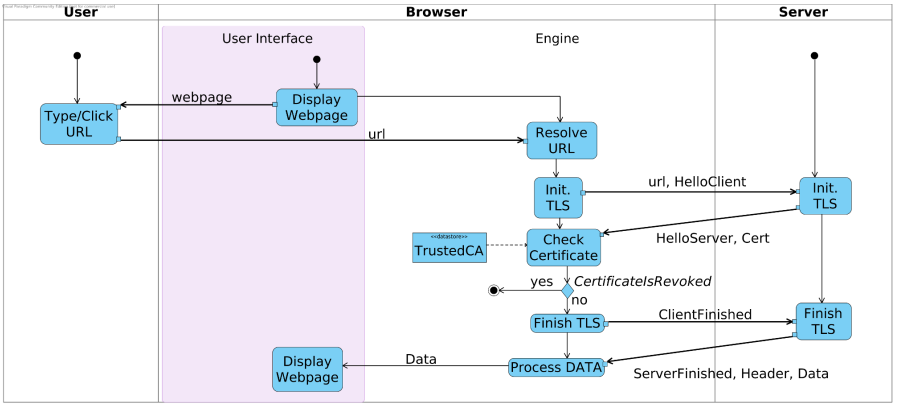

Figure 7.10: Activity diagram for certificate validation in Opera Mini

User. A user is modelled as a non-deterministic entity. This means that the user may potentially choose any of the paths of interaction that the browser offers, namely Type/Click URL or Click Button. It logically follows that this is the weaker assumption about the user skills: a ceremony that is secure for a non-deterministic user will be secure for any user. Modelling the user as a non-deterministic entity is the best approximation that we can envisage at present. A more elaborate formal model that captures the complexities of user behaviour is currently an open issue. Moreover, the feasibility of such a model is often questioned [LMM99].

Browser. The representation of the browser is split into user interface and engine. The former has the activities of Display Webpage and Display Warning. The engine normally begins with the activity Resolve URL and then starts the TLS handshake with the activity Init.TLS. The engine has the fundamental

check that concerns certificate validation, namely *CertificateIsValid*. The check follows the activity Check Certificate, which verifies if the issuer is known, the certificate is not expired, and the certificate subject matches the intended one. Moreover, the activity may verify additional information with the assistance of datastores, such as preloadedHSTSList, which stores the HSTS policy whitelist, or ServerCert, which stores the user's approved certificates. Note that the check *CertificateIsRevoked* is subject to the revocation protocol implemented by the browser. If the flow of certificate validation has not been aborted, the engine of the browser concludes a successful TLS handshake with the activity Finish TLS. Then, it runs the activity Process DATA to get the data encrypted by the server. It possibly verifies if the data contains new information about HSTS with the activity Check Header, which may lead the browser to store a new HSTS policy with the activity Store HSTS policy.

Server. We purposely consider only two activities of the server. We aim in fact to focus more on the model of the browser rather than that of the web server. The server starts the TLS handshake on its side with the activity Init.TLS, and concludes it with the activity Finish TLS. As we shall see below, a server may be corrupted by the attacker and may deviate from the supposed activity flow.

Before formally analysing the socio-technical properties on each browser, we note that the activity diagrams already offer some interesting insights. This may support the case that graphical models may be more insightful than formal ones.

As expected, SEB minimises the participation of the user also in certificate validation. In fact, Firefox, Chrome, Internet Explorer, and Safari involve the user more than SEB and Opera Mini. In particular, Firefox and Safari allow the user to store a server certificate either permanently or temporarily. However there is a fundamental difference between Firefox and Safari: the first prioritises the HSTS policy check (i.e., *URLHasHSTSpolicy*) over user's choices, the latter prioritises the user's stored server certificate (i.e., *CertificateIsStored*) over the HSTS policy. Also the activity diagrams of Opera Mini and SEB show a fundamental difference, although they look generally similar: in Opera Mini, unless the certificate is revoked, the certificate validation always leads to a successful termination of the TLS handshake; in SEB the certificate validation may instead lead to aborting the handshake according to the check *CertificateIsValid*.

Some browsers show also differences between their classic and private browsing. Firefox involves the user and implements HSTS differently in private browsing: the user cannot store a server certificate, and HSTS policies stored in earlier sessions are not considered. Also Chrome has a different implementation of HSTS in private browsing: no new HSTS policies can be permanently stored while the ones stored in previous classic sessions are considered. More surprisingly, Safari neither permanently stores HSTS policies in private browsing nor considers the ones previously stored in classic sessions.

This brief informal analysis corroborates the statement that certificate validation differs among browsers. In the next section, we see in depth by means of a formal approach how these differences affect the socio-technical security aspect of browser certificate validation.

7.6 Socio-Technical Formal Analysis

We use model checking to formally analyse certificate validation. We provide a systematic method to translate the UML activity diagrams to a formalisation in CSP# that is amenable to automatic validation by means of PAT.

From a security perspective, we define a threat model and specify the socio-technical properties that concern certificate validation in LTL.

Threat model

We consider a Man-In-The-Middle (MITM) attacker who wants to violate server authentication. He partially controls the network and can divert the browser's Init.TLS request to a corrupted server that the attacker owns. The attacker can generate a self-issued certificate, namely a new certificate signed by himself. He may alternatively have a valid certificate, namely signed by a certification authority, for a server that he controls. The attacker can interpose between the browser and the honest server that the user requests, and can replace the server certificate with one of his own. The sole limitation is that the attacker cannot sign a certificate on behalf of a certification authority.

7.6.1 Socio-Technical Security Requirements

We select five different socio-technical requirements that we deem relevant. They bind elements that span from TLS session identifiers to user choices. They aim to demonstrate how the technical mechanisms implemented in browsers inter-relate the user choices with the overall system security. We first give informal and intuitive descriptions of the requirements and then express them formally.

Definition 46 (Warning Users) *A user whose browser receives an invalid certificate is warned before the browser completes the session.*

This requirement is about a browser's warning the user that the certificate of the required server is invalid. As explained in Section 7.2, a certificate can be invalid for different reasons, each of them being more or less risky for the user. For example, some circumstances observe a server that self-issues its certificate, others conceal an attacker who attempts a MITM by injecting a fake certificate of his own.

Definition 47 (Storing Server Certificates) *A user who approves a server certificate via a browser is protected from man-in-the-middle attacks on future sessions with the same server via the same browser.*

This requirement is about how storing server certificate relates with MITM protection. When browsers receive an invalid certificate, they may still allow the user to store it. If the user assumes that a certificate in fact is trustworthy, one would expect that future sessions with the same server will be protected from MITM attacks. We shall see in the discussion that follows that this is not true for all browsers.

Definition 48 (Applying HSTS User Security) *A user who accesses a server via a browser that receives a valid certificate and an HSTS header is protected from man-in-the-middle attacks on future sessions with the same server via the same browser.*

This requirement stands on a different scenario from that of the previous one, although their conclusions are equal. This scenario sees an HSTS-compliant server who sends a valid certificate to the browser the user is using. However, it remains to be checked whether the browser is HSTS-compliant too, in which case it whitelists the server because its certificate is validated once.

Definition 49 (Applying HSTS Bootstrap) *A user who accesses a server that is pre-loaded on the browser's HSTS list is protected from man-in-the-middle attacks on future sessions with the same server via the same browser.*

This requirement concerns the relation between pre-loaded whitelist and MITM protection. In particular, we check whether HSTS-complaint browsers correctly implement the pre-loaded server whitelist to mitigate bootstrap attacks, namely MITM attacks at the first server visit.

Definition 50 (Learning from Server Certificate History) *A user who completes a TLS session with a server via a browser receiving an invalid certificate, and then completes another session with the same server via the same browser receiving a valid certificate is warned by the browser about the risk of man-in-the-middle attack.*

This last requirement aims at checking whether the browser informs the user that a MITM attack may have occurred in a *previous* TLS session. For example, if one considers a session where the browser receives an invalid certificate, then the browser may warn the user about this (according to Definition 46). If in a subsequent session the browser receives a valid certificate for the same web page, it may be the case that the former session experienced a MITM attack, hence the browser warns the user.

7.6.2 Automated Verification

Our automatic analysis relies on PAT, a model checker for the analysis of concurrent and real-time systems. Its layered design separates modelling languages from model checking algorithms, thus supporting different languages via different algorithms. It supports a range of application domains that span from bio-systems to security protocols.

PAT supports an enriched version of CSP, called CSP#. The extensions of CSP# include low-level constructs that offer a connection between data states and executable operations. Moreover, PAT supports user defined C# functions and data types that can be used directly in CSP# code as external libraries. We take advantage of this by defining an advanced data structure to model the certificate stores of browsers. PAT can model safety (i.e., bad things do not happen) and liveness (i.e., good things eventually happen) properties. In PAT, a requirement can be specified in the same language used to specify the system model, i.e., CSP, or in a temporal logic language. In our case, we use Linear Temporal Logic to specify our requirements, and resort on PAT's model checking techniques for their verification. PAT supports symbolic and explicit model checking. Since we specify our requirements in LTL, we rely on the dedicated temporal-logic model checker of PAT. In particular, we use depth-first-search as searching strategy algorithm to check whether a property is valid. If the property turns out not to be valid, we use the breadth-first-search algorithm to

find the shortest witness trace that falsifies the property. We choose to use PAT over the more popular refinement checker FDR [Ros97], because PAT supports low-level constructs written in CSP# and the specification of our requirements in LTL. However, we expect the same analysis results using another tool such as FDR.

CSP#	**Description**
$Stop$	deadlock
$Skip$	termination
$a \rightarrow P$	event prefixing
$c?[b]x \rightarrow P$	input communication
$c!e \rightarrow P$	output communication
$P[*]Q$	external choice
$P;Q$	sequential composition
$P \setminus A$	hiding
$x := e$	assignment
$if\ b\ then\ P\ else\ Q$	conditional choice
$P \mid\mid Q$	parallel composition
$P \mid\mid\mid Q$	interleaving
$P\ interrupt\ Q$	hiding

Table 7.2: CSP# syntax.

Mapping UML activity diagrams to CSP#. We systematically generate the CSP# code from UML activity diagrams, and then validate the code with PAT. Such generation is quite straightforward because we define a map between the shapes of UML activity diagrams and the CSP# syntax, which is outlined in Table 7.6.2. More precisely:

- the activity node maps to the CSP# event;
- the object (datastore) node maps to the CSP# array;
- the decision point maps to CSP# conditional choice;
- input and output objects of activities map to the CSP# values of input and output communications;
- activities roles are distinguished within a CSP# process;
- the beginning of the activity flow is a CSP# event;
- the ending of the activity flow maps to CSP# termination;
- the flow of activities within a role maps to the CSP# event prefixing;
- the flow of activities among different roles maps to CSP# input and output communications;
- the flow of data of an object node maps to the CSP# assignment.

Socio-technical requirements in LTL

PAT fully supports the LTL syntax to define a requirement about the system behaviour. An LTL formula is defined by *events*, *predefined propositions*, logical operators, and modal operators. An LTL formula can be evaluated over an infinite sequence of truth evaluations and paths. Thus, the assertion is true if every execution of the system satisfies the formula.

Although LTL defines five different modal operators, our requirements can be expressed using the combination of two operators only: $\Box$, whose semantics is that the formula holds on the current state and the entire subsequent path; $\bigcirc$, whose semantics is that the formula holds at the next state on the path. The combination $\bigcirc\Box$ expresses that the formula holds on the entire subsequent path (not necessarily in the current state).

The propositions of our LTL formulas refer to "choices" (e.g., *CertificateIsValid*) and to the activities (e.g., Init.TLS) of our UML activity diagrams. In the following definitions, we employ the same font styles used in the activity diagrams consistently. Those predicates evaluate true in states, respectively, where that choice has been selected and where that activity is executed with success. Also, our requirements include three additional LTL predicates, which we code in CSP# as macros, and one more event, namely:

- `UserWantsS` is true when a user wants to visit an honest server, namely when she types a URL or clicks a link that points to a server not corrupted by the attacker;
- `AuthFail` is true when a MITM attack succeeds, namely when the browser completes the TLS session with the attacker;
- `Preloaded` is true when an honest server is in the preloaded HSTS list of the browser;
- ServerFinished.HSTS.Data is true when the server sends that message to the browser at the end of the TLS handshake.

Assertion 1 (Warning Users)

$$\Box((\mathsf{FinishTLS} \wedge \neg\ \mathsf{DisplayWarning}) \implies \mathit{CertificateIsValid})$$

This requirement says that it is always the case that, when the browser concludes the TLS session without warnings, the certificate must have been valid.

Assertion 2 (Storing Server Certificates)

$$\Box((\mathit{CertificateIsStored} \wedge \mathit{UserWantsS} \wedge \mathsf{DisplayWebpage} \wedge \neg\ \texttt{AuthFail}) \implies \bigcirc\Box(\texttt{UserWantsS} \implies \neg\ \texttt{AuthFail}))$$

This requirement signifies that it is always the case that if the user visited a web page whose authentic certificate was stored, then the user can safely visit the web page in next sessions as the browser precludes MITM attacks.

Assertion 3 (Applying HSTS User Security)

$$\Box((\mathit{CertificateIsValid} \wedge \mathsf{ServerFinished.HSTS.Data} \wedge \mathtt{UserWantsS}) \implies \bigcirc\,\Box(\mathtt{UserWantsS} \implies \neg\,\mathtt{AuthFail}))$$

This requirement is structured as the previous requirement, and can be interpreted similarly, but it is about the HSTS policy. It says that it is always the case that if the web page is HSTS complaint, then its certificate is valid and the user can safely visit the web page in the next sessions as the browser precludes MITM attacks.

Assertion 4 (Applying HSTS Bootstrap)

$$\Box(\mathsf{Preloaded} \implies (\mathtt{UserWantsS} \implies \neg\,\mathtt{AuthFail}))$$

This requirement also relates to the HTTP Strict Transport Security (HSTS) policy. It expresses that it is always the case that if the web page is preloaded in browser's HSTS whitelist, then the user can safely visit the web page as the browser precludes MITM attacks.

Assertion 5 (Learning from Server Certificate History)

$$\Box((\mathsf{FinishTLS} \wedge \neg\ \mathit{CertificateIsValid} \wedge \mathtt{UserWantsS}) \implies \bigcirc\,\Box((\mathsf{FinishTLS} \wedge \mathit{CertificateIsValid} \wedge \mathtt{UserWantsS}) \implies \mathsf{DisplayWarning}))$$

Finally, this requirement signifies that it is always the case that if the user visited a web page whose certificate was invalid, and later she visits again the same web page but with an associated valid certificate, then the browser warns the user about the potential past MITM attack.

7.7 Findings

We studied our five requirements on the six browsers by checking the satisfiability of our formulas on the CSP# models of the certificate validation. As said above, we considered Firefox, Chrome, and Safari in three different modes: classical browsing, private browsing, and their interleaving. In total, the analysis has considered 60 different scenarios due to the mix of browsers, modes, and properties.

It was possible to encode most of the scenarios without incurring into state explosion. We needed to assume no expired certificates to avoid non termination in four scenarios that turn out to ensure the requirements: *Applying HSTS User Security* on classic browsing of Firefox, and *Applying HSTS Bootstrap* on classic browsing of Safari, Firefox, and their interleaving.

Over an Intel I7 processor running Ubuntu Linux 14.04 with 8 GB RAM, PAT 3.5.1 generally terminates the verification of each scenario in just a few seconds. However, the HSTS-related requirements take minutes on Firefox and Safari. The longest runtime is for *Applying HSTS Bootstrap* over Firefox, which takes 448 seconds. These runtimes show that our approach to socio-technical requirements is practical and can be usefully boosted further.

Interpreting the output of the tool required some effort. Table 7.7 summarises the findings. At first glance, it can be seen that the browsers that verified the highest number of requirements are SEB and Chrome, then come Internet Explorer and Firefox, and lastly Safari and Opera Mini. As a practical outcome, it turns out that an exam system that uses Safari or Opera Mini increases the chances that a user might think they are taking a valid exam when they are not.

As expected, the results of interleaving summarises the failing results of classical and private browsing: it suffices that a property fails in one of the modes (i.e., classic or private) to fail also on their interleaving. However, a more interesting result is that a session in one mode may influence a later session in a different browsing mode. This is the case of Safari for *Applying HSTS Bootstrap*. In the remainder, we comment on each requirement in detail.

Warning Users

The first requirement is found valid over SEB, Chrome, and Internet Explorer. It is also valid in Firefox in private browsing. By contrast, the model checker shows traces that falsify it over the other modes for Firefox, Safari, and Opera Mini. With Firefox and Safari the traces are similar. They report a sequence of two TLS sessions both with a MITM attack. In the first session, the browser connects to the corrupted server and warns the user, who chooses to store the certificate of the attacker anyway. In the second session, the user tries to get access to the same server, but this time the browser has the attacker server certificate stored, and completes the session without warning the user. This is due to the drawbacks of storing server certificates, which Firefox and Safari allow their users to do. Notably, Firefox in private browsing forbids the user to store server certificates, hence the requirement turns out to be valid. The trace that falsifies the property with Opera Mini is rather trivial because the browser does not involve the user at all. Opera Mini in fact shows a padlock when the certificate is valid, but even if the certificate is invalid, the browser completes the TLS session anyway, without informing the user.

Storing Server Certificate

The second property turns out to be the most tricky. It is found that all browsers verify the property except Firefox and Safari, although the latter are the only browsers that allow a user to store server certificates. This is because the property is a logical implication whose precondition is trivially falsified by the browsers that do no store server certificate. Surprisingly, the property does not hold on Firefox in classic browsing and on Safari in all modes, as they do not falsify the precondition. The user can in fact replace a server certificate as many times as she wishes to, while the browser does not inform the user that a server certificate was already stored. In support of this, the tool exhibits the following counterexample. In one session, the user engages with an honest server that transmits a self-issued certificate; the browser warns the user about the invalid certificate, but she chooses to store the certificate, thus the browser successfully concludes the TLS session. In a subsequent session, the user wants to connect again with the same server, but this time the attacker interposes and sends a self-issued certificate pretending to be the honest server; the browser

warns the user about this second invalid certificate, regardless the fact that another certificate was already stored for the same server; the user decides to store also this certificate and the browser concludes the TLS session.

Applying HSTS User Security

The third requirement is valid on SEB and in classic browsing on Firefox and Chrome. The requirement does not hold in private browsing because Firefox and Chrome trade off privacy and security: they prefer to remove HSTS policies stored during private browsing sessions to protect user's privacy. In fact an inspection of the stored HSTS policies would reveal the HSTS-complaint website the user visited in private browsing. Surprisingly, the property is not valid on Safari in any mode. The model checker shows a trace as follows: in the first session the attacker interposes in the communication, and the user chooses to store the attacker's certificate. In a subsequent session, the browser communicates with the honest server, from which it receives the HSTS header. Then, in a new session, the attacker interposes again in the communication, and the browser does not abort but concludes with no warnings. This is because a user's approved certificate bypasses the HSTS policy in Safari[2]. As expected, the property is not valid on browsers that do not support HSTS. However, it holds on SEB although it does not support HSTS, because the browser aborts when the certificate is invalid.

Applying HSTS Bootstrap

The fourth property is checked over the browsers that support HSTS. It is valid on all browsers that support HSTS except on private browsing of Safari, which does not consider the HSTS pre-loaded whitelist in private browsing[3]. Moreover, the interleaving of Safari modes also does not guarantee the property: since Safari allows the user to permanently store a certificate even in private browsing, and such storing supersedes the HSTS policy, future sessions with HSTS-complaint websites are compromised also in classic browsing.

Learning from Server Certificate History

Finally, the fifth requirement holds only on SEB, in which the precondition ($\textsc{FinishTLS} \wedge \neg$ *CertificateIsValid*) is always falsified since SEB aborts the session when the certificate is invalid. However, the property is not valid in the other browsers. This denounces the stateless philosophy whereby browsers do not record warnings they issued in the past, hence browsers cannot leverage upon them at present.

7.7.1 Recommendations

Upon the basis of our findings, we formulate the four recommendations outlined below.

[2]After we filed a bug report to Apple, we received this reply:"The information you've provided will be valuable in our efforts to determine the cause of the issue you reported."

[3]We received this reply from Apple:"Your reported issue will be addressed in upcoming releases. If you are a member of our developer program, you can test our fix in the current beta release of iOS 9 and OS X 10.11 El Capitan"

		Warning Users	Storing Server Certificate	Applying HSTS User Sec.	Applying HSTS Bootstrap	Learning from Cert. History
SEB	*co*	✓	✓	✓	−	✓
Firefox	*cb*	×	×	✓	✓	×
	pb	✓	✓	×	✓	×
	in	×	×	×	✓	×
Chrome	*cb*	✓	✓	✓	✓	×
	pb	✓	✓	×	✓	×
	in	✓	✓	×	✓	×
IE	*co*	✓	✓	×	−	×
Safari	*cb*	×	×	×	✓	×
	pb	×	×	×	×	×
	in	×	×	×	×	×
OM	*co*	×	✓	×	−	×

Note:The term *cb* indicates classic browsing, *pb* is for private browsing, and *in* is the interleaving of classic and private browsing sessions. The term *co* indicates that classic and private browsing activity diagrams coincide. The symbol ✓ indicates that the property holds; the symbol × indicates that it does not; the symbol − indicates that the property cannot be checked because the corresponding browsers do not support HSTS.

Table 7.3: The five socio-technical requirements studied over six browsers.

Recommendation 1 *Browsers should trigger users when something wrong happens rather than when something good happens.*

Browsers used to inform users when a TLS connection was going to be employed. Nowadays, most of the browsers tend to trigger and interrupt the user when some problems with TLS occur. Of course, browsers also provide positive feedback such as a locked padlock in the chrome bar, but this does not require user's participation. Opera Mini, for the sake of a lightweight implementation, shows the locked padlock as a positive feedback but does not warn users in case of problems with TLS.

Recommendation 2 *Browsers should consider the preloaded whitelist of HSTS-complaint servers also in private browsing.*

Firefox and Chrome show that it is possible to protect the user and mitigate bootstrap attacks with HSTS without breaking user's privacy in private browsing. The choice of Safari not to consider the preloaded whitelist leads to some weaknesses that when mixed with other features (i.e., user approved server certificates) may become serious vulnerabilities.

Recommendation 3 *HSTS policies should have priority over user's past choices.*

Also this recommendation comes from the findings on Safari, which implements a customised HSTS mechanism. HSTS has been conceived to avoid user's participation on security choices. A server that chooses to be HSTS-complaint cannot self-issue a certificate. Thus, any check on user's approved certificates should be superseded by the HSTS policy stored in the browser.

Recommendation 4 *Browsers should keep track of invalid certificate history to warn users more appropriately.*

Users may forget past security warnings, and browsers may help. For example, browsers could maintain a cache of invalid certificate hashes. In doing so, it would be possible for browsers to warn users when a different invalid certificate is presented by a server with which the browser communicated in the past. It is worth noting that looking at past interactions is the strategy that the Session Description Protocol [Len06] advances to strengthen the management of self-issued certificates. Surprisingly, it has not been used in HTTPS.

7.8 Conclusion

Since humans play critical roles in an exam, the security of such protocol should consider their participation. We observe that browsers are the main component that nowadays interfaces the user with the network, not only in exam protocols, but also in many other security protocols.

The socio-technical analysis of the security of browsers is yet to be considered innovative at present. It combines traditional analysis of the technologies underlying browsers on the one hand, with elements of user participation on the other. By doing so, the socio-technical approach is oriented at characterising security requirements also in terms of what the user may accomplish, with the ultimate aim of building browsers that are secure *in* the presence of humans.

This chapter describes our work in this area. It focuses on server authentication with the user via the browser. More specifically, it studies the socio-technical ceremony of certificate validation in the various circumstances where this validation can fail, including MITM attacks.

The security analysis of the ceremony of certificate validation from a socio-technical standpoint inspires a number of research questions, and we concentrated on three: the first addresses the differences in terms of user participation in server authentication; the second concerns the strategies that browsers use to reduce the security risks for users in the presence of an invalid certificate; the third relates to how browsers improve the security by involving the users more profitably than they do at present.

To address these questions we formulated five requirements that tackle how users are involved in the ceremony of certificate validation. The outcome of our analysis demonstrates that each browser implements the ceremony of certificate validation differently, and that this is the origin of a few security problems. In particular, our analysis shows that HSTS fails on its goal when implemented in the wide customised process of certificate validation. Microsoft announced that the next version of Internet Explorer will support HSTS [Mic15]. We argue that its usefulness will depend on how HSTS is implemented in the browser's certificate validation.

A major hallmark throughout our work is the adoption of UML activity diagrams as a semi-formal language to represent portions of browser functioning compactly, so that the human analyser could quickly get to grasps with their niceties. However, rigorous security analysis requires a formal approach. Thus, the diagrams that represent the ceremonies are systematically mapped into CSP# and validated in the PAT model checker against the LTL specification of our five socio-technical requirements. These are the main steps of our approach to the socio-technical formal analysis of the security of browsers. The current findings encourage us to develop this approach further, for example by automating it fully, and by trying it out on additional socio-technical requirements.

It is worth to stress that our current choices of formal languages and supporting tools are not meant to be binding; rather, they aim at demonstrating our approach. Also, we are currently working on reproducing the experiments described above on different tools. We advocate alternative verification methods to check other requirements such as privacy, a requirement that cannot be modelled in PAT. While looking at the interleaving of sessions in different browser modes, we noted that a session in private browsing should not interfere the following sessions in classic browsing. A different approach, possibly a different model checker such as FDR [Ros97], is required to understand whether this interference may leak information, since PAT cannot verify privacy requirements. We leave this further development as future work.

Chapter 8

Conclusions

The main argument of this dissertation is that an exam must be designed and analysed as carefully as security protocols are. Thus, a rigorous understanding of the security requirements of exams is fundamental to design protocols that withstand threats coming from malicious candidates and authorities. Moreover, it is necessary to develop formal approaches that allow one to prove that an exam protocol meets the stated security requirements. We address these needs using formal methods for the security analysis of exams, and cryptographic techniques for their design.

This dissertation has found that secure exam should meet ten fundamental requirements, five concerning authentication and five concerning privacy. Authentication requirements aim to preserve the association between candidate identity, mark, and test throughout the entire examination. Privacy requirements aim to provide anonymity to both candidates and examiners. This dissertation has also shown that exam should provide verifiability. Similarly to the end-to-end verifiability requirement in voting, having verifiability in any exam is fundamental to raise the credibility of the exam process. Thus, we have identified six individual and five universal verifiability requirements. Exam protocols should provide enough information to allow candidates and auditors to verify the correctness of the exam using private and public available information. In summary, this dissertation has achieved Objective 1, as specified in chapter 1, by advancing 21 security requirements for exams.

This research work has also provided novel approaches that allow the study of security requirements for exams. The formal framework for the analysis of authentication and privacy enables the evaluation of exam protocols, which can be specified in the applied π–calculus. The framework is flexible as it supports various types of exams, namely traditional, computer-assisted, and Internet-based exams. The formal framework for the analysis of verifiability consists of an abstract model that supports a wide choice of verification methods based either on symbolic or on computational models. Both frameworks can be extended with additional requirements. In fact, we formulate the extra requirements of Notification Request Authentication and Dispute Resolution. Overall, this dissertation has introduced two formal frameworks for the analysis of exam protocols, hence it has met Objective 2.

The scarcity of secure exam protocols in the literature, has led us to propose three new protocols for traditional, computer-assisted, and Internet-based

exam. The protocols exploit different cryptographic techniques that span from exponentiation mixnet to visual cryptography, but share the same design principle of minimising the reliance on the trusted third parties. In this vein, we have proposed two protocol versions of existing exam software (WATA). Using our formal frameworks, it has been possible to analyse the security of existing and proposed exam protocols. Some security issues have been found and modification have been suggested. The three novel protocols and the two protocol versions of exam software contribute to the achievement of Objective 3.

It has been observed that the human should be also considered for an end-to-end security analysis of exams. Moreover, browsers are often the main components of an exam protocol and are critical to exam security as they interface users and machines. Thus, this dissertation has advanced a socio-technical analysis of certificate validation as carried out in a browser for secure exams (SEB). It has proposed a method that allows us to expand the analysis to the most popular browsers. The method consists of modelling the certificate validation ceremonies using UML Activity Diagram, and of the systematic translation of such diagrams to CSP processes. We have validated the method with the model checking of five socio-technical requirements that binds TLS session, certificates and user choices. Thus, this dissertation has achieved Objective 4 by exploring a socio-technical understanding of the certificate validation in exam and modern browsers.

This research sees interesting outcomes, and we recall them. One is about the formal analysis of the existing exam protocols: it is observed that authentication can fail because of the presence of logic flaws in the design of the protocols, while privacy can fail because the protocols rely upon inadequate cryptographic primitives. A second insight is about the relation between exams and similar domains, such as voting. Although the approaches to design and analysis are similar, we find that several requirements are different. Another interesting outcome is the design of cryptographic exam protocols that meet contrasting requirements without the need of a TTP and in presence of remote and face-to-face phases. Exams should not rely on TTP and should guarantee some forms of accountability. One last notable result derives from the socio-technical analysis of certificate validation ceremonies. It turns out that the security of such ceremonies varies considerably when implementors customise the ceremonies, possibly including users in security decisions.

As any research, this work required us to make some choices. One is about the classes of security requirements. We considered the formalisation of authentication, privacy, and verifiability, but definitions of non-repudiation and accountability are also relevant for exams. Our formalisation of Dispute Resolution goes in that direction. Another choice is about the list of considered requirements. We deemed the elements of this list highly desirable according to our experience and discussions with colleagues. However, the list is not meant to be exhaustive: some exams may demand additional requirements, as is the case of WATA protocols.

Our analysis inherits the limitations of ProVerif. Although the analysis is sound, ProVerif implements safe abstractions that might lead to non termination or false attacks. We overcome some limitations with manual proofs, as in the case of the analysis of Remark! for universal verifiability.

In summary, this work has identified the security requirements for exams, developed frameworks for their analysis, and designed new protocols. The re-

sults of this research offer a promising foundation for the design and analysis of secure and practical protocols that accommodate any type of exam. The next section outlines a number of possible research directions for this area.

8.1 Future Work

Future work that continues the work presented in this dissertation can be envisaged over different research directions.

Concerning the formal frameworks, it might be possible to extend them with the specification of new security requirements and to study formally the relation between the proposed requirements. Also, the analysis of more exam protocols would help to corroborate the flexibility of the proposed frameworks. In particular, it is interesting to analyse the verifiability of exams in the computational model, possibly with the assistance of the CryptoVerif [Bla08] automatic tool. Another interesting research direction is to study whether our approach in defining verifiability can be applied to e-voting.

Regarding Remark!, future work includes the extension of open-source platforms like Moodle with our protocol. Another interesting research direction is to expand Remark! with techniques to detect plagiarism and candidate cheating at testing. This is a significant point since Remark! is designed to allow candidates to take the exam from home. We envisage that misbehaviour detection strategies such as data mining used to derive patterns described by Pieczul and Foley [PF14] can be useful for this purpose. Another research direction includes the support for collaborative marking, in which the questions are categorised by subject, and examiners evaluate only the answers that pertain to the examiner subject area.

Future work can be envisaged also for computer-assisted exam protocols. One is to extend the proposed protocols to accommodate different exam scenarios. For instance, some scenarios may not require the participation of the candidate at notification. To achieve this, we envisage a temporal deanonymization solution similar to the one specified in chapter 5 for Remark!. We note that Dispute Resolution may conflict with the anonymity of the candidate's test. Thus it would be interesting to study how we can get both of them in the design of exam protocols, and in general how to ensure both accountability and privacy requirements in the same system.

Another line of research concerns the formal analysis. It might be possible to study compositional proofs that integrate computational proofs of the cryptographic primitives used in our protocol with the symbolic ones obtained in ProVerif. A practical research direction is the implementation of a prototype of the protocol and the verification of whether different visual cryptography schemes can be used to increase the perceptual security of an exam.

The socio-technical analysis of certification validation shows that SEB is a promising browser, so it would be interesting to integrate it with Remark! to achieve secure testing. Finally, the socio-technical analysis approach can be developed further at least in two directions: one is to fully automate the translation from UML Activity Diagram to CSP#; the other is to model additional socio-technical requirements, perhaps considering privacy.

Bibliography

[AAVS13] Devdatta Akhawe, Bernhard Amann, Matthias Vallentin, and Robin Sommer. Here's my cert, so trust me, maybe?: Understanding tls errors on the web. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 59–70. International World Wide Web Conferences Steering Committee, 2013.

[ABB+05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P. C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The avispa tool for the automated validation of internet security protocols and applications. In *Proceedings of the 17th International Conference on Computer Aided Verification*, CAV'05, pages 281–285, Berlin, Heidelberg, 2005. Springer-Verlag.

[ABJB10] Gaurav Aggarwal, Elie Bursztein, Collin Jackson, and Dan Boneh. An analysis of private browsing modes in modern browsers. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[ABL+10] Devdatta Akhawe, Adam Barth, Peifung E. Lam, John Mitchell, and Dawn Song. Towards a formal foundation of web security. pages 290–304, 2010.

[ABR12] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Privacy supporting cloud computing: Confichair, a case study. In *Proceedings of the First International Conference on Principles of Security and Trust*, POST'12, pages 89–108. Springer-Verlag, Berlin, Heidelberg, 2012.

[ABR13] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Privacy-supporting cloud computing by in-browser key translation. *J. Comput. Secur.*, 21(6):847–880, November 2013.

[ACC+08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra. Formal analysis of saml 2.0 web browser single sign-on: Breaking the saml-based single sign-on for google apps. In *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering*, FMSE '08, pages 1–10, New York, NY, USA, 2008. ACM.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, pages 104–115, New York, NY, USA, 2001. ACM.

[AF13] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 257–272, Washington, D.C., 2013. USENIX.

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. pages 36–47, 1997.

[AMRR14] Myrto Arapinis, Loretta Ilaria Mancini, Eike Ritter, and Mark Ryan. Privacy through pseudonymity in mobile telephony systems. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[AST13] Islam Abdelhalim, Steve Schneider, and Helen Treharne. An integrated framework for checking the behaviour of fuml models using csp. *International Journal on Software Tools for Technology Transfer*, 15(4):375 – 396, August 2013.

[BAN90] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, February 1990.

[BBS13] M.L. Bolton, E.J. Bass, and R.I. Siminiceanu. Using formal verification to evaluate human-automation interaction: A review. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(3):488–503, May 2013.

[BCCKR11] Giampaolo Bella, Gianpiero Costantino, Lizzie Coles-Kemp, and Salvatore Riccobene. Remote management of face-to-face written authenticated though anonymous exams. In *CSEDU*, pages 431–437. SciTePress, 2011.

[BCK12] Giampaolo Bella and Lizzie Coles-Kemp. Layered Analysis of Security Ceremonies. In *Information Security and Privacy Resevarch SE-23*, volume 376 of *IFIP Advances in Inf. and Communication Technology*, pages 273–286. Springer-Verlag, 2012.

[BCR10] Giampaolo Bella, Gianpiero Costantino, and Salvatore Riccobene. Wata - a system for written authenticated though anonymous exams. In *CSEDU (2)'10*, pages 132–137, 2010.

[Ben96] Josh Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, December 1996.

[BGL13] Giampaolo Bella, Rosario Giustolisi, and Gabriele Lenzini. What security for electronic exams? In *International Conference on Risks and Security of Internet and Systems (CRiSIS)*, October 2013.

[BGW01] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: The insecurity of 802.11. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, pages 180–189, New York, NY, USA, 2001. ACM.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, pages 90–101, New York, NY, USA, 2009. ACM.

[BHM08] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*, CSF '08, pages 195–209, Washington, DC, USA, 2008. IEEE Computer Society.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, pages 82–, Washington, DC, USA, 2001. IEEE Computer Society.

[Bla08] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, Oct 2008.

[BMU08] Michael Backes, Matteo Maffei, and Domenique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *IEEE Symposium on Security and Privacy, 2008. SP 2008.*, pages 202–215, May 2008.

[BRT13] Josh Benaloh, Peter Y.A. Ryan, and Vanessa Teague. Verifiable postal voting. In *Security Protocols XXI*, volume 8263 of *Lecture Notes in Computer Science*, pages 54–65. Springer Berlin Heidelberg, 2013.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 544–553, New York, NY, USA, 1994. ACM.

[BvOP+09] Robert Biddle, P. C. van Oorschot, Andrew S. Patrick, Jennifer Sobey, and Tara Whalen. Browser interfaces and extended validation ssl certificates: An empirical study. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 19–30, New York, NY, USA, 2009. ACM.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, SFCS '85, pages 372–382, Washington, DC, USA, 1985. IEEE Computer Society.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, February 1981.

[CJS+07] Iliano Cervesato, AaronD. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Christopher Walstad. Breaking and fixing public-key kerberos. In *Advances in Computer Science - ASIAN 2006. Secure Software and Related Issues*, volume 4435 of *Lecture Notes in Computer Science*, pages 167–181. Springer Berlin Heidelberg, 2007.

[CLN09] Cas J.F. Cremers, Pascal Lafourcade, and Philippe Nadeau. Comparing state spaces in automatic security protocol analysis. In *Formal to Practical Security*, volume 5458 of *Lecture Notes in Computer Science*, pages 70–94. Springer Berlin Heidelberg, 2009.

[CRHJDJ06] Jordi Castellà-Roca, Jordi Herrera-Joancomartí, and Aleix Dorca-Josa. A secure e-exam management system. In *ARES*, pages 864–871. IEEE Computer Society, 2006.

[CS14] Chris Culnane and Steve Schneider. A peered bulletin board for robust use in verifiable voting systems. In *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*, pages 169–183, July 2014.

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, 2008.

[CvO13] J. Clark and P.C. van Oorschot. Sok: Ssl and https: Revisiting past challenges and evaluating certificate trust model enhancements. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 511–525, May 2013.

[DHL13] J. Dreier, Jonker H., and P. Lafourcade. Defining verifiability in e-auction protocols. In *ASIACCS'13*, pages 547–552. ACM, 2013.

[DJL13] Jannik Dreier, Hugo Jonker, and Pascal Lafourcade. Defining verifiability in e-auction protocols. In *ASIACCS*, pages 547–552. ACM, 2013.

[DJP10] Naipeng Dong, Hugo L. Jonker, and Jun Pang. Analysis of a receipt-free auction protocol in the applied pi calculus. In *FAST'10*, volume 6561 of *LNCS*. Springer, 2010.

[DKR06] Stephanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proceedings of the 19th IEEE Workshop on Computer Security Foundations*,

CSFW '06, pages 28–42, Washington, DC, USA, 2006. IEEE Computer Society.

[DKR09] Stephanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, jul 2009.

[DLL12] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. A formal taxonomy of privacy in voting protocols. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6710–6715, June 2012.

[DLL13] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. Formal verification of e-auction protocols. In David Basin and JohnC. Mitchell, editors, *Principles of Security and Trust*, volume 7796 of *Lecture Notes in Computer Science*, pages 247–266. Springer Berlin Heidelberg, 2013.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, 2008.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[ECHA09] Aleks Essex, Jeremy Clark, Urs Hengartner, and Carlisle Adams. How to print a secret. In *USENIX Conference on Hot Topics in Security*, HotSec. USENIX Association, 2009.

[EDTLR01] Michael Elkins, David Del Torto, Raph Levien, and Thomas Roessler. MIME Security with OpenPGP. RFC 3156, 2001.

[EKOT14] Keita Emura, Akira Kanaoka, Satoshi Ohta, and Takeshi Takahashi. Building secure and anonymous communication channel: formal model and its prototype implementation. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 1641–1648. ACM, 2014.

[Elg85] Tareq Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Ell07] Carl Ellison. Ceremony design and analysis. *IACR eprint archive*, pages 1–17, 2007.

[F99] F. Javier Thayer Fábrega. Strand spaces: Proving security protocols correct. *J. Comput. Secur.*, 7(2-3):191–230, March 1999.

[FJ95] Simon N. Foley and Jeremy L. Jacob. Specifying security for computer supported collaborative working. *Journal of Computer Security*, 3:233–253, 1995.

[FL05] Scott Flinn and Jo Lumsden. User perceptions of privacy and security on the web. In *Third Annual Conference on Privacy, Security and Trust, October 12-14, 2005, The Fairmont Algonquin, St. Andrews, New Brunswick, Canada, Proceedings*, 2005.

[FOK+98] Steven Furnell, P. D. Onions, Martin Knahl, Peter W. Sanders, Udo Bleimann, U. Gojny, and H. F. Röder. A security framework for online distance learning and training. *Internet Research*, 8(3):236–242, 1998.

[GFZN09] Nataliya Guts, Cdric Fournet, and Francesco Zappa Nardelli. Reliable evidence: Auditability by typing. In Michael Backes and Peng Ning, editors, *Computer Security ESORICS 2009*, volume 5789 of *Lecture Notes in Computer Science*, pages 168–183. Springer Berlin Heidelberg, 2009.

[GIJ+12] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: Validating ssl certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 38–49, New York, NY, USA, 2012. ACM.

[GJ03] Philippe Golle and Markus Jakobsson. Reusable anonymous return channels. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, WPES '03. ACM, 2003.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20, 1982.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[GMSS08] Sebastian Gajek, Mark Manulis, Ahmad-Reza Sadeghi, and Jörg Schwenk. Provably secure browser-based user-aware mutual authentication over tls. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '08, pages 300–311, New York, NY, USA, 2008. ACM.

[Gol96] Dieter Gollmann. What do we mean by entity authentication? In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 46–, Washington, DC, USA, 1996. IEEE Computer Society.

[Gop07] Saravanan Gopinathan. Globalisation, the singapore developmental state and education policy: a thesis revisited. *Globalisation, Societies and Education*, 5(1):53–70, 2007.

[GPS05] Thomas Groß, Birgit Pfitzmann, and Ahmad-Reza Sadeghi. Browser model for security analysis of browser-based protocols. In *Proceedings of the 10th European Conference on Research in Computer Security*, ESORICS'05, pages 489–508, Berlin, Heidelberg, 2005. Springer-Verlag.

[HCZ15] Feng Hao, Dylan Clarke, and Avelino Francisco Zorzo. Deleting secret data with public verifiability. *Dependable and Secure Computing, IEEE Transactions on*, PP(99):1–1, 2015.

[HJB12] Jeff Hodges, Collin Jackson, and Adam Barth. HTTP Strict Transport Security (HSTS). RFC 6797, 2012.

[HM05] Changhua He and John C. Mitchell. Security analysis and improvements for IEEE 802.11i. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005, San Diego, California, USA*, 2005.

[HMPs14] Susan Hohenberger, Steven Myers, Rafael Pass, and abhi shelat. Anonize: A large-scale anonymous survey system. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 375–389, Washington, DC, USA, 2014. IEEE Computer Society.

[Hoa78] Charles A. R. Hoare. Communicating Sequential Processes. *Commun. ACM*, 21(8):666–677, 1978.

[HP10] Andrea Huszti and Attila Petho. A secure electronic exam system. *Publicationes Mathematicae Debrecen*, 77(3-4):299–312, 2010.

[HPC04] Jordi Herrera-Joancomartí, Josep Prieto-Blázquez, and Jordi Castellà-Roca. A secure electronic examination protocol using wireless networks. In *International Conference on Information Technology: Coding and Computing (ITCC'04), Volume 2, April 5-7, 2004, Las Vegas, Nevada, USA*, pages 263–268, 2004.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, pages 539–556, Berlin, Heidelberg, 2000. Springer-Verlag.

[HS11] Rolf Haenni and Olivier Spycher. Secure internet voting on limited devices with anonymized dsa public keys. In *WOTE'11*. USENIX, 2011.

[HS12] Paul Hoffman and Jakob Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, 2012.

[JVRK12] Audun Josang, Kent A. Varmedal, Christophe Rosenberger, and Rajendra Kumar. Service provider authentication assurance. In *Tenth Annual International Conference on Privacy, Security and Trust (PST)*, pages 203–210. IEEE Computer Society, July 2012.

[KER09] Michael Kazin, Rebecca Edwards, and Adam Rothman. *The Princeton Encyclopedia of American Political History. (Two volume set)*. The Princeton Encyclopedia of American Political History. Princeton University Press, 2009.

[KLP14] Sudeep Kanav, Peter Lammich, and Andrei Popescu. A conference management system with verified document confidentiality. In *Computer Aided Verification (CAV)*. 2014.

[KPS10] Dan Kaminsky, Meredith L Patterson, and Len Sassaman. PKI layer cake: new collision attacks against the global x.509 infrastructure. In *Proceedings of the FC'10*, pages 289–303, Berlin, Heidelberg, 2010. Springer-Verlag.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *Proceedings of the 15th European Conference on Research in Computer Security*, ESORICS'10, pages 389–404, Berlin, Heidelberg, 2010. Springer-Verlag.

[KuTV10] Ralf K üsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and relationship to verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 526–535, New York, NY, USA, 2010. ACM.

[Len06] Jonathan Lennox. Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP). RFC 4572, 2006.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate Transparency. RFC 6962, 2013.

[LMM99] Diego Latella, Istvan Majzik, and Mieke Massink. Towards a formal operational semantics of uml statechart diagrams. In *Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, pages 465–, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.

[Low96] Gavin Lowe. Breaking and fixing the needham-schroeder publickey protocol using fdr. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, TACAs '96, pages 147–166, London, UK, UK, 1996. Springer-Verlag.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, CSFW '97, pages 31–, Washington, DC, USA, 1997. IEEE Computer Society.

[Mar09a] Moxie Marlinspike. More tricks for defeating ssl in practice. *DEFCON 17*, 2009.

[Mar09b] Moxie Marlinspike. New tricks for defeating ssl in practice. *BlackHat DC, February*, 2009.

[MDRH14] Alex Migicovsky, Zakir Durumeric, Jeff Ringenberg, and J.Alex Halderman. Outsmarting proctors with smartwatches: A case study on wearable computing security. In *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 89–96. Springer Berlin Heidelberg, 2014.

[Mea96] Catherine Meadows. Language generation and verification in the nrl protocol analyzer. In *Computer Security Foundations Workshop, 1996. Proceedings., 9th IEEE*, pages 48–61, Jun 1996.

[Mer83] Michael John Merritt. *Cryptographic Protocols*. PhD thesis, Atlanta, GA, USA, 1983.

[Mou09] Jean-Pierre Moussette. Method for anonymous computerized processing of documents or object. Patent, 08 2009. EP 1430439 B1.

[MPR13] Matteo Maffei, Kim Pecina, and Manuel Reinert. Security and privacy by declarative design. In *Computer Security Foundations Symposium (CSF)*, pages 81–96. IEEE, 2013.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–40, September 1992.

[MR94] Sjouke Mauw and Michel A. Reniers. An algebraic semantics of basic message sequence charts. *The Computer Journal*, 37(4):269–277, 1994.

[Nee02] Roger M. Needham. Back to the beginning. In *Security Protocols, 10th International Workshop, Cambridge, UK, April 17-19, 2002, Revised Papers*, page 242, 2002.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[NS95] Moni Naor and Adi Shamir. Visual cryptography. In *Advances in Cryptology EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 1995.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, January 1998.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO*, LNCS, pages 129–140. Springer, 1992.

[PF14] Olgierd Pieczul and Simon N. Foley. Collaborating as normal: detecting systemic anomalies in your partner. In *Proceedings of 22nd International Security Protocols Workshop, Cambridge*, 2014.

[Res00] Eric Rescorla. HTTP Over TLS. RFC 2818, 2000.

[Ros97] Andrew W. Roscoe. *The Theory and Practice of Concurrency. international series in computer science.* Prentice-Hall, 1997.

[RS00] Peter Y.A. Ryan and Steve Schneider. *The modelling and analysis of security protocols: the csp approach.* Addison-Wesley Professional, first edition, 2000.

[RS01] Peter Y. A. Ryan and Steve A. Schneider. Process algebra and non-interference. *Journal of Computer Security*, 9(1-2):75–103, January 2001.

[RS06] Peter Y. A. Ryan and Steve A. Schneider. Prêt à voter with re-encryption mixes. In *Proceedings of the 11th European conference on Research in Computer Security*, ESORICS'06, pages 313–326. Springer-Verlag, 2006.

[RS11] Mark D. Ryan and Ben Smyth. Applied pi calculus. In *Formal Models and Techniques for Analyzing Security Protocols*, chapter 6. IOS Press, 2011.

[RT10] Blake Ramsdell and Sean Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751, 2010.

[SBL+12] Daniel R. Schneider, Dirk Bauer, Marco Lehre, Thomas Piendl, and Benno Volk. The safe exam browser: Innovative open source software for online examinations. *18th International Conference on Technology Supported Learning & Training*, 2012.

[Sch98] Steve Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, Sep 1998.

[SEA+09] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, pages 399–416, Berkeley, CA, USA, 2009. USENIX Association.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SLDP09] Jun Sun, Yang Liu, JinSong Dong, and Jun Pang. Pat: Towards flexible verification under fairness. In *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer Berlin Heidelberg, 2009.

[SMA+13] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, 2013.

[SRKM10] Ben Smyth, Mark Ryan, Steve Kremer, and K. Mounira. Towards automatic analysis of election verifiability properties. In *ARSPA-WITS'10*, volume 6186 of *LNCS*, pages 146–163. Springer, 2010.

[SS96] Steve Schneider and Abraham Sidiropoulos. Csp and anonymity. In *Computer Security ESORICS 96*, volume 1146 of *Lecture Notes in Computer Science*, pages 198–218. Springer Berlin Heidelberg, 1996.

[Tze04] Wen-Guey Tzeng. Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters. *IEEE Transactions on Computers*, 53(2):232–240, 2004.

[vDMR08] Tony van Deursen, Sjouke Mauw, and Sasa Radomirović. Untraceability of RFID protocols. In *Information Security Theory and Practices. Smart Devices, Convergence and Next Generation Networks (WISTP'08)*, volume 5019 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[VFBH13] Nevena Vratonjic, Julien Freudiger, Vincent Bindschaedler, and Jean-Pierre Hubaux. The Inconvenient Truth About Web Certificates. In *Economics of Information Security and Privacy III*, pages 79–117. Springer New York, 2013.

[Wei05] Edgar Weippl. *Security in E-learning*, volume 6 of *Advances in Information Security*. Springer Science + Business Media, 2005.

[WL93] Thomas .Y.C. Woo and Simon S. Lam. A semantic model for authentication protocols. In *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on*, pages 178–194, May 1993.

[Yee13] Peter E. Yee. Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 6818, 2013.

[YP13] Li Yuan and Steven Powell. MOOCs and Open Education: Implications for Higher Education: A White Paper. Technical report, JICS - Centre for Educational Technology & Interoperable Standards (CETIS), 2013.

[YSA05] Zishuang (Eileen) Ye, Sean Smith, and Denise Anthony. Trusted paths for browsers. *ACM Transactions on Information and System Security*, 8(2):153–186, May 2005.

[Zim92] Phil Zimmermann. PGP user's guide, 1992.

Website Bibliography

[Ald15] Aldo Cortesi. mitmproxy 0.12. https://mitmproxy.org, 2015.

[Bur12] Elie Bursztein. 19% of users use their browser private mode. http://www.elie.net/blog/privacy/19-of-users-use-their-browser-private-mode, 2012.

[Com15] Questionmark Computing. Questmark Secure. https://www.questionmark.com/content/questionmark-secure/, August 2015.

[Cop13] Larry Copeland. School cheating scandal shakes up atlanta. http://www.usatoday.com/story/news/nation/2013/04/13/atlanta-school-cheatring-race/2079327/, April 2013.

[Cou12] Sean Coughlan. Harvard and MIT online courses get 'real world' exams. http://www.bbc.com/news/education-19505776, September 2012.

[Cou15] Coursera. Earn a Course Certificate. https://www.coursera.org/signature/, August 2015.

[ETS15] ETS. Tests and Products. https://www.ets.org/tests_products/alpha, July 2015.

[Fid15] Fidenia. QuestBase. http://www.questbase.com/, August 2015.

[Fig14] Le Figaro. Le concours de médecine, une sélection impitoyable. http://etudiant.lefigaro.fr/les-news/actu/detail/article/le-concours-de-medecine-une-selection-impitoyable-5428/, May 2014.

[Ger15] Gerald Combs. Wireshark 1.12. https://www.wireshark.org, 2015.

[Giu15] Rosario Giustolisi. The code used to verify the case studies of this dissertation. https://sites.google.com/site/sarogiustolisi/cabinet/thesiscode.tar.gz, 2015.

[Goo12] Google. CRLSets. https://dev.chromium.org/Home/chromium-security/crlsets, 2012.

[Inc15] ProctorU Inc. ProctorU - Online Proctoring. http://www.proctoru.com, August 2015.

[INF15] INFOSAFE. Anonymous Marking. http://www.anonymousmarking.com/, July 2015.

[Kar15] Karl von Randow. Charles 3.10. http://www.charlesproxy.com, 2015.

[Lan12] Adam Langley. SSL interstitial bypass rates. https://www.imperialviolet.org/2012/07/20/sslbypassrates.html, 2012.

[Lev04] Adine Levine. Grading the Curve at HLS. http://hlrecord.org/?p=11168, November 2004.

[Lew13] Tamar Lewin. Students Rush to Web Classes, but Profits May Be Much Later. http://www.nytimes.com/2013/01/07/education/massive-open-online-courses-prove-popular-if-not-lucrative-yet.html?_r=0, January 2013.

[Lip14] Kevin Liptak. U.S. Navy discloses nuclear exam cheating. http://edition.cnn.com/2014/02/04/us/navy-cheating-investigation/, February 2014.

[LSV15] LSV. Security Protocols Open Repository (SPORE). http://www.lsv.ens-cachan.fr/Software/spore/, 2015.

[Mar15] Moxie Marlinspike. Convergence. http://convergence.io/, 2015.

[Mic15] Microsoft. HTTP Strict Transport Security comes to Internet Explorer. http://blogs.msdn.com/b/ie/archive/2015/02/16/http-strict-transport-security-comes-to-internet-explorer.aspx, 2015.

[Mim14] Michael Mimoso. IE 12 to support hsts encryption protocol. https://threatpost.com/ie-12-to-support-hsts-encryption-protocol/105266, 2014.

[Min14] Ministère de l'éducation nationale de l'enseignement supérieur et de la recherche. Statistiques du concours général des lycées. http://eduscol.education.fr/cid49775/organisation-du-concours-general-des-lycees.html, November 2014.

[MP13] Moxie Marlinspike and Trevor Perrin. Trust Assertions for Certificate Keys. http://tools.ietf.org/html/draft-perrin-tls-tack-02, 2013.

[Neo15] Neoptec. Nemo Scan: the reliable anonymity system. http://www2.neoptec.com/en/products/nemoscan/, July 2015.

[New15] BBC News. Vyapam: India's deadly medical school exam scandal. http://www.bbc.com/news/world-asia-india-33421572, July 2015.

[Off13] European Personnel Selection Office. Careers with the european union. http:\europa.eu/epso/apply/how_apply/index_en.htm, November 2013.

[OMG11] OMG. Unified Modeling Language Superstructure. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF, 2011.

[Ope14] Opera Mini. State of the Mobile Web, 2014.

[oS15] The University of Sheffield. Marking policies and procedures. https://www.sheffield.ac.uk/scharr/current/pgt/marking, July 2015.

[Pea15] Pearson. Pearson VUE delivered exams. http://home.pearsonvue.com/test-taker/All-Tests.aspx, July 2015.

[Pre14] Associated Press. Navy kicks out 34 for nuke test cheating. http://www.foxnews.com/us/2014/08/21/navy-kicks-out-34-for-nuke-cheating/, August 2014.

[Sch15] Stanford Law School. Exams and Papers. http://www.law.stanford.edu/organizations/offices/office-of-the-registrar/exam-information, July 2015.

[Sta15] StatCounter. Top 5 Browsers. http://gs.statcounter.com/#browser-ww-monthly-201411-201501, 2015.

[The11] The Chromium Blog. New Chromium security features. blog.chromium.org/2011/06/new-chromium-security-features-june.html, 2011.

[TOE] Test of English as a Foreign Language. http://www.ets.org/toefl.

[Uni15] Dublin City University. Anonymous Marking. http://www4.dcu.ie/iss/am/index.shtml, July 2015.

[U.S15] U.S. Army. Army Knowledge Online. https://akologin.us.army.mil, 2015.

[Wat14] Robert Watson. Student visa system fraud exposed in BBC investigation. http://www.bbc.com/news/uk-26024375, February 2014.

Publications

[BCGL14] Giampaolo Bella, Paul Curzon, Rosario Giustolisi, and Gabriele Lenzini. A socio-technical methodology for the security and privacy analysis of services. In *2014 IEEE 38th International Computer Software and Applications Conference Workshops (COMPSACW)*, pages 401–406, July 2014.

[BGL13a] Giampaolo Bella, Rosario Giustolisi, and Gabriele Lenzini. Socio-technical formal analysis of TLS certificate validation in modern browsers. In *Eleventh Annual International Conference on Privacy, Security and Trust, PST 2013, 10-12 July, 2013, Tarragona, Catalonia, Spain, July 10-12, 2013*, pages 309–316. IEEE, 2013.

[BGL13b] Giampaolo Bella, Rosario Giustolisi, and Gabriele Lenzini. A socio-technical understanding of tls certificate validation. In *Trust Management VII*, volume 401 of *IFIP Advances in Information and Communication Technology*, pages 281–288. Springer Berlin Heidelberg, 2013.

[BGL13c] Giampaolo Bella, Rosario Giustolisi, and Gabriele Lenzini. What security for electronic exams? In *International Conference on Risks and Security of Internet and Systems (CRiSIS)*, October 2013.

[BGL14] Giampaolo Bella, Rosario Giustolisi, and Gabriele Lenzini. Secure exams despite malicious management. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*, pages 274–281, July 2014.

[BGLR15] Giampaolo Bella, Rosario Giustolisi, Gabriele Lenzini, and Peter Y.A. Ryan. A secure exam protocol without trusted parties. In *ICT Systems Security and Privacy Protection*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 495–509. Springer International Publishing, 2015.

[DGK+14a] Jannik Dreier, Rosario Giustolisi, Ali Kassem, Pascal Lafourcade, and Gabriele Lenzini. On the verifiability of (electronic) exams. Technical Report TR-2014-2, 2014.

[DGK+14b] Jannik Dreier, Rosario Giustolisi, Ali Kassem, Pascal Lafourcade, Gabriele Lenzini, and Peter Y. A. Ryan. Formal analysis of electronic exams. In *SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, August*, pages 101–112. SciTePress, 2014.

[DGK+15] Jannik Dreier, Rosario Giustolisi, Ali Kassem, Pascal Lafourcade, and Gabriele Lenzini. A framework for analyzing verifiability in traditional and electronic exams. In *Information Security Practice and Experience*, volume 9065 of *Lecture Notes in Computer Science*, pages 514–529. Springer International Publishing, 2015.

[FGH+13] Ana Ferreira, Rosario Giustolisi, Jean-Louis Huynen, Vincent Koenig, and Gabriele Lenzini. Studies in socio-technical security analysis: Authentication of identities with tls certificates. In *Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '13, pages 1553–1558, Washington, DC, USA, 2013. IEEE Computer Society.

[GLR14] Rosario Giustolisi, Gabriele Lenzini, and Peter Y.A. Ryan. Remark!: A secure protocol for remote exams. In *Security Protocols XXII*, volume 8809 of *Lecture Notes in Computer Science*, pages 38–48. Springer International Publishing, 2014.

Appendix A

CSP# Code

A.1 Specification of Common Parts

```
#import "PAT.Lib.Set";
#import "PAT.Lib.SetMine";

//UML activities' objects and certificate's fields
enum {HelloClient, HelloServer, ClientFinished, ServerFinished,
      Data, Warning, Webpage, Continue, Abort, StoreCertificate,
      Pk, HSTS, No_HSTS, S, I, SignCA, SignS, SignI, expi,
      noexpi, revo, norevo};

channel ui 0;
channel network 0;

//UML datastores, certificate, and typed/clicked url
var<Set> dynamicHSTSList;
var<Set> preloadedHSTSList;
var<SetArray> ServerCert;
var cert[3];
var extendedcert[6];
var typed_url: {S..I}=S;

//UML decision points
#define CertificateIsValid cert[0]==typed_url &&
                           cert[2]==SignCA &&
                           extendedcert[4]==noexpi;

#define CertificateIsValidNR cert[0]==typed_url &&
                             cert[2]==SignCA &&
                             extendedcert[4]==noexpi &&
                             extendedcert[5]==norevo;
                             //This used by Safari
#define URLhasHSTSpolicy dynamicHSTSList.Contains(typed_url) ||
                         preloadedHSTSList.Contains(typed_url);
```

```
#define CertificateIsStored ServerCert.Contains(extendedcert);

//Variables to keep track of some session's event
var intruder_server=false;
var user_warned=false;
var finishTLS=false;
var preload=false;

//Intruder process chooses which server plays session by session//
Intruder()= ServerI() [] ServerH();

//-----Intruder server process-----//
ServerI() = []header:{HSTS, No_HSTS}@ []url:{S,I}@
            []sk:{SignI,SignCA}@
            Init_TLS ->
            network?urlx.HelloClient ->
            //Intruder cannot sign certificate on behalf of CA
            ifa (url==S && sk==SignCA) {
               network!HelloServer.url.Pk.SignI -> Skip}
            else {network!HelloServer.url.Pk.sk -> Skip};
            Finish_TLS ->
            network?m  ->
            ifa (m==ClientFinished) {
               INTRUDER_IN{intruder_server=true} ->
               network!ServerFinished.header.Data ->Skip
            };
            Intruder();

//-----Honest server process-----//
ServerH() = []header:{HSTS, No_HSTS}@  []sk:{SignS,SignCA}@
            Init_TLS ->
            network?urlx.HelloClient ->
            network!HelloServer.S.Pk.sk ->
            Finish_TLS ->
            network?m ->
            ifa (m==ClientFinished) {
               network!ServerFinished.header.Data ->Skip};
            Intruder();

//-----User process-----//
User() =    ui?webpage ->
            case {
            //The user can type or click on either honest's or
            //intruder's url.
            webpage == Webpage: ui!S{typed_url=S} -> User() []
                                ui!I{typed_url=I} -> User()
            webpage == Warning: ui!StoreCertificate -> User()[]
                                ui!Continue -> User() []
                                ui!Abort -> User()
            default: User()};
```

```
//------Model process-----//
Model = Preloading() [] Begin();
Preloading = PreloadHSTSpolicy->{preloadedHSTSList.Add(S);
                                 preload=true} -> Begin;
Begin = Intruder() ||| User() ||| Browser();

//User who wants to visit the honest server
#define UserwantS typed_url==S;
//Successful MITM attack: User wants to visit the honest server,
//but browser completed with the intruder
#define AuthFail intruder_server && UserwantS;
#define User_warned user_warned;
#define CompleteTLS finishTLS;
#define Preload preload;

///---------Properties--------///
#assert Model deadlockfree;
//Property 1
#assert Model |=[] ((CompleteTLS && !User_warned) ->
                    CertificateIsValid);

//Property 2
#assert Model |=[]((CertificateIsStored && UserwantS &&
                   ui.Data && !AuthFail)->
                   X([](UserwantS -> !AuthFail)));

//Property 3
#assert Model |=[]((CertificateIsValid &&
                   network.ServerFinished.HSTS.Data && UserwantS)->
                   X([](UserwantS -> !AuthFail)));

//Property 4
#assert Model |=[] (Preload-> (UserwantS -> !AuthFail));


//Property 5
#assert Model |=[]((CompleteTLS && !CertificateIsValid &&
                    UserwantS)->
                  X([]((CompleteTLS && CertificateIsValid &&
                        UserwantS)->
                       User_warned)));
```

A.2 Specification of Web Browsers

SEB

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
                       user_warned=false;} ->
            ui?url ->
            Resolve_URL ->
            Init_TLS ->
            network!url.HelloClient ->
            network?HelloServer.id.pk.sk{cert[0]=id;cert[1]=pk;
                                         cert[2]=sk} ->
            Check_Certificate ->
            if (rev==revo) {{finishTLS=false} -> Skip}
            else {
                if (CertificateIsValid) {{finishTLS=true}->Skip}
                else {  {finishTLS=false} -> Skip }
            };
            if (!finishTLS)
             //The browser informs the server about Abort (sync)
             {network!Abort -> Skip}
            else {
                Finish_TLS ->
                network!ClientFinished ->
                Process_DATA ->
                network?ServerFinished.header.Data ->
                Display_Webpage ->
                ui!Data -> Skip
            };
            Browser();
```

Firefox

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
                       user_warned=false;} ->
            ui?url ->
            Resolve_URL ->
            Init_TLS ->
            network!url.HelloClient ->
            network?HelloServer.id.pk.sk{extendedcert[0]=cert[0]=id;
                                         extendedcert[1]=cert[1]=pk;
                                         extendedcert[2]=cert[2]=sk;
                                         extendedcert[3]=url;
                                         extendedcert[4]=exp} ->
```

```
            Check_Certificate ->
            ifa (CertificateIsValid ) {{finishTLS=true} -> Skip}
            else {
                ifa (URLhasHSTSpolicy || rev==revo)
                   {{finishTLS=false} -> Skip}
                else {
                    ifa (CertificateIsStored)
                       {{finishTLS=true} -> Skip}
                    else {
                        DisplayWarning ->
                        ui!Warning{user_warned=true} ->
                        ui?userchoice ->
                        tau{
                            if (userchoice == Abort)
                              {finishTLS=false}
                            else {
                                finishTLS=true;
                                if (userchoice == StoreCertificate)
                                  {ServerCert.Add(extendedcert);}
                                  //associates a url to
                                  //the server certificate
                            }
                        } -> Skip
                    }
                }
            };
            ifa (!finishTLS)
             //The browser informs the server about Abort (sync)
              {network!Abort -> Skip}
            else {
                Finish_TLS ->
                network!ClientFinished ->
                Process_DATA ->
                network?ServerFinished.header.Data ->
                Display_Webpage ->
                ui!Data ->
                Check_Header ->
                ifa (header==HSTS && CertificateIsValid)
                   {StoreHSTSpolicy->
                    {dynamicHSTSList.Add(cert[0])} -> Skip}
            };
            Browser();
```

Firefox - Private browsing

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
                       user_warned=false;} ->
```

```
            ui?url ->
            Resolve_URL ->
            Init_TLS ->
            network!url.HelloClient ->
            network?HelloServer.id.pk.sk{extendedcert[0]=cert[0]=id;
                                         extendedcert[1]=cert[1]=pk;
                                         extendedcert[2]=cert[2]=sk;
                                         extendedcert[3]=url;
                                         extendedcert[4]=exp} ->
            Check_Certificate ->
            if (CertificateIsValid) {{finishTLS=true} -> Skip}
            else {
                if (URLhasHSTSpolicy || rev==revo)
                  {{finishTLS=false} -> Skip}
                else {
                    if (CertificateIsStored)
                      {{finishTLS=true} -> Skip}
                    else {
                        DisplayWarning ->
                        ui!Warning{user_warned=true} ->
                        ui?userchoice ->
                        tau{
                            if (userchoice == Abort)
                              {finishTLS=false}
                            else {
                                finishTLS=true;
                            }
                        } -> Skip
                    }
                }
            };
            if (!finishTLS)
            //The browser informs the server about Abort (sync)
            {network!Abort -> Skip}
            else {
                Finish_TLS ->
                network!ClientFinished ->
                Process_DATA ->
                network?ServerFinished.header.Data ->
                Display_Webpage ->
                ui!Data -> Skip
            };
            Browser();
```

Chrome

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
```

```
            user_warned=false;expc=exp} ->
        ui?url ->
        Resolve_URL ->
        Init_TLS ->
        network!url.HelloClient ->
        network?HelloServer.id.pk.sk{cert[0]=id;cert[1]=pk;
                                     cert[2]=sk} ->
        Check_Certificate ->
        if (CertificateIsValid) {{finishTLS=true} -> Skip}
        else {
            if (URLhasHSTSpolicy || rev==revo)
              {{finishTLS=false} -> Skip}
            else {
                DisplayWarning ->
                ui!Warning{user_warned=true} ->
                ui?userchoice ->
                tau{
                    if (userchoice == Abort)
                      {finishTLS=false}
                    else { finishTLS=true; }
                } -> Skip
            }
        };
        if (!finishTLS)
         //The browser informs the server about Abort (sync)
          {network!Abort -> Skip}
        else {
            Finish_TLS ->
            network!ClientFinished ->
            Process_DATA ->
            network?ServerFinished.header.Data ->
            Display_Webpage ->
            ui!Data ->
            Check_Header ->
            if (header==HSTS && CertificateIsValid)
              {StoreHSTSpolicy->
               {dynamicHSTSList.Add(cert[0])} -> Skip}
        };
        Browser();
```

Chrome - Private browsing

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
                       user_warned=false;expc=exp} ->
            ui?url ->
            Resolve_URL ->
            Init_TLS ->
```

```
            network!url.HelloClient ->
            network?HelloServer.id.pk.sk{cert[0]=id;cert[1]=pk;
                                         cert[2]=sk} ->
            Check_Certificate ->
            if (CertificateIsValid) {{finishTLS=true} -> Skip}
            else {
                if (URLhasHSTSpolicy || rev==revo)
                  {{finishTLS=false} -> Skip}
                else {
                    DisplayWarning ->
                    ui!Warning{user_warned=true} ->
                    ui?userchoice ->
                    tau{
                        if (userchoice == Abort)
                          {finishTLS=false}
                        else { finishTLS=true; }
                    } -> Skip
                }
            };
            if (!finishTLS)
             //The browser informs the server about Abort (sync)
              {network!Abort -> Skip}
            else {
                Finish_TLS ->
                network!ClientFinished ->
                Process_DATA ->
                network?ServerFinished.header.Data ->
                Display_Webpage ->
                ui!Data -> Skip
            };
            Browser();
```

Safari

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
            user_warned=false;} ->
            ui?url ->
            Resolve_URL ->
            Init_TLS ->
            network!url.HelloClient ->
            network?HelloServer.id.pk.sk{extendedcert[0]=cert[0]=id;
                                         extendedcert[1]=cert[1]=pk;
                                         extendedcert[2]=cert[2]=sk;
                                         extendedcert[3]=url;
                                         extendedcert[4]=exp;
                                         extendedcert[5]=rev} ->
            Check_Certificate ->
```

```
            if (CertificateIsValidNR || CertificateIsStored)
              {{finishTLS=true} -> Skip}
            else {
                if (URLhasHSTSpolicy || rev==revo)
                  {{finishTLS=false} -> Skip}
                else {
                    DisplayWarning ->
                    ui!Warning{user_warned=true} ->
                    ui?userchoice ->
                    tau{
                        if (userchoice == Abort)
                        {finishTLS=false}
                        else {
                            finishTLS=true;
                            if (userchoice == StoreCertificate)
                              {ServerCert.Add(extendedcert);}
                              //associates a url to
                             //the server certificate
                        }
                    } -> Skip
                }
            };
            if (!finishTLS)
            //The browser informs the server about Abort (sync)
            {network!Abort -> Skip}
            else {
                Finish_TLS ->
                network!ClientFinished ->
                Process_DATA ->
                network?ServerFinished.header.Data ->
                Display_Webpage ->
                ui!Data ->
                Check_Header ->
                if (header==HSTS && CertificateIsValidNR)
                  {StoreHSTSpolicy->
                   {HSTSList.Add(cert[0])} -> Skip}
            };
            Browser();
```

Safari - Private browsing

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
                       user_warned=false;} ->
            ui?url ->
            Resolve_URL ->
            Init_TLS ->
            network!url.HelloClient ->
```

```
            network?HelloServer.id.pk.sk{extendedcert[0]=cert[0]=id;
                                         extendedcert[1]=cert[1]=pk;
                                         extendedcert[2]=cert[2]=sk;
                                         extendedcert[3]=url;
                                         extendedcert[4]=exp;
                                         extendedcert[5]=rev} ->
            Check_Certificate ->
            if (CertificateIsValidNR || CertificateIsStored)
              {{finishTLS=true} -> Skip}
            else {
                DisplayWarning ->
                ui!Warning{user_warned=true} ->
                ui?userchoice ->
                tau{
                    if (userchoice == Abort) {finishTLS=false}
                    else {
                        finishTLS=true;
                        if (userchoice == StoreCertificate)
                          {ServerCert.Add(extendedcert);}
                        //associates a url to the server certificate
                    }
                } -> Skip
            };
            if (!finishTLS)
            //The browser informs the server about Abort for syncing
            {network!Abort -> Skip}
            else {
                Finish_TLS ->
                network!ClientFinished ->
                Process_DATA ->
                network?ServerFinished.header.Data ->
                Display_Webpage ->
                ui!Data -> Skip
            };
            Browser();
```

Internet Explorer

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
                       user_warned=false;expc=exp} ->
            ui?url ->
            Resolve_URL ->
            Init_TLS ->
            network!url.HelloClient ->
            network?HelloServer.id.pk.sk{cert[0]=id;cert[1]=pk;
                                         cert[2]=sk} ->
            Check_Certificate ->
```

```
            if (rev==revo) {{finishTLS=false} -> Skip}
            else {
                if (CertificateIsValid) {{finishTLS=true} -> Skip}
                else {
                   DisplayWarning ->
                   ui!Warning{user_warned=true} ->
                   ui?userchoice ->
                   tau{
                       if (userchoice == Abort) {finishTLS=false}
                       else {finishTLS=true; }
                   } -> Skip
                }
            };
            if (!finishTLS)
            //The browser informs the server about Abort (sync)
            {network!Abort -> Skip}
            else {
                Finish_TLS ->
                network!ClientFinished ->
                Process_DATA ->
                network?ServerFinished.header.Data ->
                Display_Webpage ->
                ui!Data -> Skip
            };
            Browser();
```

Opera Mini

```
Browser() = []rev:{revo, norevo}@[]exp:{expi,noexpi}@
            Display_Webpage ->
            //New session, variables used in macros are reset
            ui!Webpage{finishTLS=false; intruder_server=false;
                       user_warned=false;expc=exp} ->
            ui?url ->
            Resolve_URL ->
            Init_TLS ->
            network!url.HelloClient ->
            network?HelloServer.id.pk.sk{cert[0]=id;cert[1]=pk;
                                         cert[2]=sk} ->
            Check_Certificate ->
            if (rev==revo) {{finishTLS=false} -> Skip}
            else {{finishTLS=true} -> Skip};
            if (!finishTLS)
            //The browser informs the server about Abort (sync)
            {network!Abort -> Skip}
            else {
                Finish_TLS ->
                network!ClientFinished ->
                Process_DATA ->
                network?ServerFinished.header.Data ->
```

```
    Display_Webpage ->
    ui!Data -> Skip
};
Browser();
```