UNIVERSITÉ GRENOBLE ALPES

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathematiques & Informatique**

Présentée par

Ali KASSEM

Thèse dirigée par **Yassine LAKHNECH**
et codirigée par **Pascal LAFOURCADE**

préparée au sein **du Laboratoire Verimag**
et de **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Automated Verification of Exam, Cash, Reputation, and Routing Protocols

Thèse soutenue publiquement le **18 September 2015**,
devant le jury composé de :

Dr. Steve KREMER
Inria Nancy – Grand Est, LORIA, Président

Dr. Sébastien GAMBS
Université de Rennes 1 – INRIA, Rapporteur

Pr. Olivier PEREIRA
Université catholique de Louvain, Rapporteur

Pr. Luca VIGANÒ
King's College London, Examinateur

Pr. Yassine LAKHNECH
Université Joseph Fourier, Directeur de thèse

Dr. Pascal LAFOURCADE
Université Joseph Fourier, Co-Directeur de thèse

The author was employed by the Université Joseph Fourier, a part of the Université Grenoble Alpes. The work is prepared at the laboratory Verimag, with the support of the doctoral school Mathématiques, Sciences et Technologies de l'Information, Informatique (MSTII).

Supervisor:
Prof. Yassine Lakhnech (Université Joseph Fourier, Verimag).

Daily Advisor:
Dr. Pascal Lafourcade (Université Joseph Fourier, Verimag).

Abstract

Security is a crucial requirement in the applications based on information and communication technology, especially when an open network such as the Internet is used. To ensure security in such applications several security protocols have been developed. However, the design of complex security protocols is notoriously difficult and error-prone. Several flaws have been found on protocols that are claimed secure. Hence, security protocols must be verified before they are used. One approach to verify security protocols is the use of formal methods. The use of formal methods has enabled the discovery of several flaws on security protocols, as well as, the proof of some other protocols' correctness. However, errors can be introduced when the protocols are implemented. Another approach which can be used to verify implementations individual executions is runtime verification. Runtime verification mainly can help in the cases where verifying implementations formally is complex and difficult.

In this thesis we contribute to security protocol verification with an emphasis on formal verification and automation. Firstly, we study exam protocols. We propose formal definitions for several authentication and privacy properties in the Applied π-Calculus. We also provide an abstract definitions of verifiability properties. We analyze all these properties automatically using ProVerif on multiple case studies, and identify several flaws. Moreover, we propose several monitors to check exam requirements at runtime. These monitors are validated by analyzing a real exam implementation using MarQ tool. Secondly, we propose a formal framework to verify the security properties of non-transferable electronic cash protocols. We define client privacy and forgery related properties. Again, we illustrate our model by analyzing three case studies using ProVerif, and we re-discover known attacks. Thirdly, we propose formal definitions for authentication, privacy, and verifiability properties of electronic reputation protocols. We discuss the proposed definitions, with the help of ProVerif, on a simple reputation protocol. Finally, we obtain a reduction result to verify route validity of ad-hoc routing protocols in presence of multiple independent attackers.

Keywords: Authentication, Privacy, Verifiability, Formal Verification, RunTime Verification, Exams, Electronic Cash, Electronic Reputation, Route Validity, Applied π-Calculus, ProVerif, QEA, MarQ.

Résumé

La sécurité est une exigence cruciale dans les applications basées sur l'information et la technologie de communication, surtout quand un réseau ouvert tel que l'Internet est utilisé. Pour assurer la sécurité dans ces applications nombreux protocoles cryptographiques ont été développé. Cependant, la conception de protocoles de sécurité est notoirement difficile et source d'erreurs. Plusieurs failles ont été trouvées sur des protocoles prétendus sécurisé. Par conséquent, les protocoles cryptographiques doivent être vérifiés avant d'être utilisés. Une approche pour vérifier les protocoles cryptographiques est l'utilisation des méthodes formelles. L'utilisation des méthodes formelles a permis la découverte de plusieurs failles sur les protocoles de sécurité, ainsi que la preuve de la justesse de certains autres protocoles. Toutefois, des erreurs peuvent être introduites lorsque les protocoles sont mis en œuvre. Une autre approche qui peut être utilisée pour vérifier les implémentations exécutions individuelles est la vérification de l'exécution. Vérification de l'exécution principalement peut aider dans les cas où la vérification des implémentations formellement est complexe et difficile.

Dans cette thèse, nous contribuons à la vérification des protocoles cryptographiques avec un accent sur la vérification formelle et l'automatisation. Tout d'abord, nous étudions les protocoles d'examen. Nous proposons des définitions formelles pour plusieurs propriétés d'authentification et de confidentialité dans le π-calcul appliqué. Nous fournissons également des définitions abstraites des propriétés de vérifiabilité. Nous analysons toutes ces propriétés en utilisant automatiquement ProVerif sur plusieurs études de cas, et avons identifié plusieurs failles. En outre, nous proposons plusieurs moniteurs pour vérifier les exigences d'examen à l'exécution. Ces moniteurs sont validés par l'analyse d'un exécutions d'examens réels en utilisant l'outil MarQ. Deuxièmement, nous proposons un cadre formel pour vérifier les propriétés de sécurité des protocoles de monnaie électronique. Nous définissons la notion de vie privée du client et les propriétés de la falsification. Encore une fois, nous illustrons notre modèle en analysant trois études de cas à l'aide ProVerif, et nous redécouvrons les attaques connues. Troisièmement, nous proposons des définitions formelles des propriétés de l'authentification, la confidentialité et le vérifiabilité des protocoles de réputation électronique. Nous discutons les définitions proposées, avec l'aide de ProVerif, sur un protocole simple de réputation. Enfin, nous obtenons un résultat de réduction quand vous cherchez des attaques sur la route validité en présence de plusieurs attaquants indépendants qui ne partagent pas leurs connaissances.

Mots-clés : Authentification, La confidentialité, La vérifiabilité, La vérification formelle, La vérification de l'exécution, Examens, Argent électronique, La réputation électronique, Route validité, Appliqué π-calcul, ProVerif, QEA, MarQ.

Acknowledgements

Firstly I would like to thank all the members of my jury, in particular my reviewers Dr. Sébastien Gambs and Pr. Olivier Pereira, for accepting to report on this long manuscript. I would also like to express my gratitude towards the examiners Dr. Steve Kremer and Pr. Luca Viganò for their interest in my work. I am also very grateful to my supervisors Pr. Yassine Lakhnech and Dr. Pascal Lafourcade for accepting me as a PhD student, and supervising me. I have always enjoyed working with them, and without them this work would not have been possible.

Then I would like to thank my collaborators, Jannik Dreier from Institute of Information Security – ETH Zurich for his help on using ProVerif, and his collaboration on the formal verification of exam and cash protocols; Rosario Giustolisi, Gabriele Lenzini, and Peter Y. A. Ryan from SnT/University of Luxembourg for their collaboration on the formal verification of exam protocols; and Yliès Falcone from "Université Grenoble Alpes" for his help and collaboration on exam monitoring. I also would like to thank Stéphane Devismes for his help and ideas on route validity for routing protocols; François Géronimi from THEIA, Daniel Pagonis from TIMC-IMAG, and Olivier Palombi from LJK for providing us with a description of e-exam software system at "Université Joseph Fourier", and for sharing with us the logs of some real french e-exams; Andrea Huszti and Attila Pethő for the helpful discussions on our findings concerning their exam protocol; Claire Maiza for her help on structuring and organizing this manuscript; and Giles Reger from University of Manchester for answering my questions, and for providing me with help on using the MarQ tool.

I also want to thank Sebastian Mödersheim from DTU Informatics - Denmark, and all the colleges at VERIMAG in particular the director Nicolas Halbwachs, the secretaries (Sandrine Magnin, Christine Saunier, Rosen Carbonero) for their help with the paperwork, and the IT-Team (Philippe Genin, Patrick Fulconis, ...).

Finally I am very grateful to my family and friends for their constant support. Last but not least my thanks go to all those I forgot here.

"Essentially, all models are wrong, but some are useful."

George Edward Pelham Box.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the development and spread of the technology more and more electronic applications have emerged, and more and more services are offered over the Internet. For instance, several universities and other educational organizations (*e.g.*, Coursera[1], CISCO[2]) offer computer/Internet-based exams. Banking and electronic payment systems such as smart cards or, *e.g.*, PayPal[3] allow electronic payments and the online transfer of the money around the world. Several sites allow the customers to leave feedbacks which reflect their satisfaction (*e.g.*, Amazon[4], eBay[5], Yelp[6]). To implement such distributed applications several protocols have been developed. These protocols use cryptography to ensure security properties, such as secrecy and authentication. However, the design of complex secure protocols is error-prone. Hence, it is necessary to verify these protocols before implementing them. The main goal of this thesis is to propose formal models and definitions to express security properties and verify cryptographic protocols.

In this chapter we motivate our work, and provide a general overview of related works. Then, we present the structure of the thesis and our contributions. Finally, we list previous publications concerning the results presented in this thesis.

Contents

[1] www.coursera.org [2] www.cisco.com [3] www.paypal.com [4] www.amazon.com [5] www.ebay.com [6] www.yelp.com

1.1 Cryptographic Protocols

Cryptographic protocols are distributed algorithms that make use of cryptography to achieve some security properties over a public network, such as the Internet. Examples of basic security properties that are required by most Cryptographic protocols are secrecy and message authentication.

- Secrecy: The attacker cannot compute or obtain the value of a secret. A stronger variant of this property (strong secrecy) states that an attacker cannot distinguish when the value of the secret changes (going back to Goldwasser and Micali [GM82]). It ensures that the attacker cannot obtain any information about the secret.

- Message authentication (or data origin authentication): This property ensures that a message has not been modified while in transit (data integrity) and that the recipient can verify the source of the message. To this end some protocols use primitives such as digital signatures which provide an evidence that a message was not modified and was sent by someone possessing the proper signing key. Properties such as message authentication can be modeled as correspondence assertions [RSG+00, RS11]: "on every execution trace an event $\mathtt{e}_2$ is preceded by an event $\mathtt{e}_1$".

However, up-to-date protocols such as electronic exam, cash, and reputation protocols require more complex properties like privacy properties, which include:

- Anonymity: The ISO/IEC standard 15408 [ISO12] defines anonymity as ensuring that a user may use a service or resource without disclosing the user's identity. This definition is close to what is called anonymity in terms of unlinkability by Pfitzmann *et al.* [PK00]. In this sense, anonymity hides the link between a user's identity and a service or a resource rather than hiding the identity itself. For instance, sender anonymity says that the attacker cannot link a sender to a given message, and student anonymity says that the attacker cannot link a student to a given answer.

- Untraceability: The attacker cannot tell whether two actions are made by the same participant. For example, a bank cannot trace two payments to the same user, or an attacker cannot link two sessions which involve the same RFID tag.

- Receipt-Freeness: A participant cannot prove to the attacker that he takes a certain action. For example, a customer cannot prove (provide a receipt) to a coercer that he provided a certain feedback.

Other interesting properties include verifiability, that is a user can check that a certain information is correct. An example of verifiability properties is to verify the eligibility of users attempting to use a service or a resource. This requires authenticating the users

and verifying that they are authorized to use the service or the resource. Moreover, some protocols require specific security properties. For instance, validity of the obtained route is a crucial property that has to be ensured by a routing protocol. An important property that is specific to electronic cash protocols is the ability to identify the client that spends a coin twice.

There are two main classes of security properties: trace (reachability) and equivalence (indistinguishability) properties. Trace properties are defined on every execution trace of the protocol. Primary examples of trace properties are secrecy and authentication properties. For instance, secrecy is expressed as a state where the secret is revealed cannot be reached on every execution trace of the protocol. Verifiability properties can be expressed using *verification tests* [DJL13, KRS10a]. A verification test is an algorithm or a logical formula that has to satisfy certain conditions with respect to the related property. Verifiability properties can be expressed as (un)reachability properties [DJL13, SRKK10]. Route validity can also be expressed as a reachability property [ACD10]. Equivalence properties are defined as equivalences between protocol executions. Two executions are equivalent if the attacker cannot distinguish between them. Equivalence properties include strong secrecy and other privacy properties (*e.g.*, anonymity, untraceability, and receipt-freeness).

A natural question is: how we can prove that a cryptographic protocol ensures a given security property?

The flaws (*e.g.*, [MSS98, DKS10, ACC+08, HSD+05, CS11]) found over the recent 20 years show that designing security protocols is error-prone. Hence, provided that security failures can have serious financial and privacy consequences, cryptographic protocols verification is a necessity.

1.2 Symbolic Verification of Cryptographic Protocols

Due to the difficulty of designing secure protocols in general, as we argued before, several flaws were discovered on cryptographic protocols, even without breaking the underlying cryptography. For instance, the attack on Needham-Schroeder protocol works without breaking any of the used cryptographic primitives. Instead, it requires two parallel executions of the protocol with an attacker in the middle of two honest participants. Hence, proving the security of cryptographic primitives is not sufficient, and verification in needed to argue that a protocol is secure or correct.

An approach to verify cryptographic protocols is the use of formal methods. Formal methods provide mathematically based techniques for modeling and verifying cryptographic protocols. They allow to detect the shortcomings in protocols design, and give evidence of security within a given model if no flaws are found. Moreover, in case where a

flaw is found they might provide a counter-example which gives insight to fix the analyzed protocol.

In formal verification, we can distinguish two main approaches: the symbolic and the computational approaches. In symbolic approach, abstracted models such as constraint systems [MS01], process algebra (*e.g.*, [Hoa85, AF01]), and dedicated logics (*e.g.*, [BAN90, Bla01]) are used to represent a protocol and a security property. Then, the protocol is checked whether it satisfies the property or not, for instance, by exploring the state space searching if an invalid state can be reached. Whereas in computational approach, probabilistic arguments are used to analyze the security of protocols. Typically, a security property is defined as a game that involves an attacker. A game corresponds to an attack scenario for which the advantage of the attacker can be computed. If the advantage of the attacker is not negligible, he breaks the property. In this thesis, we adopt the symbolic approach to verify security properties of exam, reputation, and cash protocols. Moreover, we provide a reduction result for verifying route validity in wireless sensor networks. In the following, we give an overview of symbolic models and tools, and the successful results of their use in verification of cryptographic protocols.

In the symbolic model, messages are represented by terms, and cryptographic primitives such as encryption, signature, hash, etc. are abstracted as black boxes and assumed to be perfectly secure. Such a model was first proposed by Dolev and Yao [DY81, DY83a] who developed the idea of an attacker that has full control of the communication network and unbounded computational power, but can apply only some specific predefined rules on the messages in his knowledge. In addition to Dolev and Yao model the symbolic approach encompasses a variety of techniques including: logics such as the BAN-logic [BAN90] used to model authentication and Horn clauses [Bla01], process algebras such as the Communicating Sequential Processes [Hoa85] and the Applied π-Calculus [AF01], and typing methods such as F7 [BFG10, BBF+11] and F* [SCF+13] for verifying protocols implementations.

Several flaws were found by using symbolic methods to verify security protocols. For example, Mitchell *et al.* [MSS98] found anomalies in Secure Socket Layer (SSL) 3.0 using finite-state analysis, Delaune *et al.* [DKS10] discovered several attacks on the PKCS 11 standard which defines an API to cryptographic tokens, Armando *et al.* [ACC+08] found a server security flaw on Google's Single Sign-On protocol, the attack allows a dishonest service provider to impersonate a user at another service provider. Several weaknesses have been identified by Cortier and Smyth [CS11] on Helios 2.0 voting system [ADMPQ09], and Dreier *et al.* [DLL13] discovered several attacks on auction protocols due to Brandt [Bra06] and Curtis *et al.* [CPS07]. Formal methods can also be used to prove the absence of attacks in a formal model under certain assumptions. For example, He *et al.* [HSD+05] carried out a modular proof of IEEE 802.11i and Transport

Layer Security (TLS) using a Protocol Composition Logic (PCL). However, the main challenges in protocols verification remain: (i) in the choice of the model to represent the protocol and the security properties, (ii) the formalization of security properties as informal definitions tend to contain imprecisions, (iii) and the development of tools to automatically verify security protocols as human proofs tend to be error-prone as complexity increase.

The main goal of this thesis is to propose formal models and definitions to express security properties and verify security protocols. We propose models and definitions for privacy, authentication and verifiability properties of electronic exam (e-exam) protocols (also suitable for pencil-and-paper exams). Privacy can mean secrecy of the exam questions, anonymity of a student that submits a given answer, or anonymity of an examiner that attributes a given mark. Authentication includes authenticating the student that submits an answer, and verifiability includes verifying that all answers are accepted from registered candidates. Then, we study electronic cash (e-cash) and propose formal definitions for client privacy (ensuring that the attacker can link neither a client to a coin he spend nor two coins spent by the same client) and forgery related properties such as unforgability, and double spending identification. We also propose a formal framework for security analysis of electronic reputation (e-reputation) protocols. Again we consider privacy, authentication and verifiability properties. We also discuss several case studies to validate our models on existing security protocols. We re-discover some known flaws, and we also discover several new flaws on the analyzed protocols. Moreover, we generalize the reduction result for analyzing route validity presented in [CDD12] to the case of multiple independent attackers, which do not share their knowledge.

1.2.1 Attacker Model

Flaws on the security of protocols highly depend on the capabilities of the attacker. For instance, considering too restrictive attacker model can lead to missing some flaws. For example, Needham-Schroeder protocol was proven secure using BAN logic for a single protocol execution, while the flaw found by G. Lowe requires two parallel executions of the protocol between the attacker and two participants. On the other hand, too powerful attacker can lead to unrealistic attacks which are difficult to be applied in practice. For example, when analyzing routing protocols of wireless sensor networks it is not realistic to assume that the attacker controls all the communications. However he has to be located somewhere in the network, and thus can control only a finite number of nodes [ACD10].

In the symbolic model the cryptography is assumed to be perfect, and the attacker is usually assumed to has a full control of the (public) network, such as the Dolev-Yao attacker [DY81, DY83a]. The Dolev-Yao attacker can eavesdrop, remove, substitute, duplicate and delay messages that the parties are sending one another, and insert messages

of his choice on the public channels. However, he can manipulate messages, *e.g.*, extract data from messages or compose new messages from known data, only under the assumption of perfect cryptography. For instance, the only way to decrypt an encryption $\texttt{enc}(m, k)$ of a message m with a key k is to know the corresponding inverse key $\texttt{inv}(k)$, where `enc` is an encryption function and `inv` is a key inverse function. Provided that `dec` is the corresponding decryption function of `enc`, the latter feature can be expressed by the equation $\texttt{dec}(\texttt{enc}(m, k), \texttt{inv}(k))$ which captures the notion that the encryption function `enc` is perfect. In the case where the encryption function has some additional features, *e.g.*, if it is a homomorphic function, then it is important to have equations that capture such features. Otherwise, some attacks might be missed. Note that, usually a set of equations or a set of inference deduction rules is defined to captures all the capabilities of the attacker.

In this thesis, we adopt a Dolev-Yao like attacker to analyze the security properties of exam, cash and reputation protocols. Additionally we consider corrupted parties to model *e.g.*, bribing and coercion. Whereas for analyzing the security of routing protocols in wireless sensor networks, we consider multiple independent local attackers that do not share their knowledge. We assume that each attacker has Dolev-Yao like capabilities, but compromises only one node and thus only controls the communications between this node and its neighbors.

1.2.2 Automation

To this end, several tools [ABB+05, Bla01, Cre08a, Cre08b, SMCB12, MSCB13, CcCK12] have been developed to support automated protocol verification in symbolic model. Note, as protocol verification problems are in general undecidable [EGS85, DLM04]–in particular with unbounded message length and an unbounded number of parallel instances– these tools employ techniques such as approximations, restrictions to bounded number of instances and limited attacker capabilities, or they do not always terminate. In the following, we summarize some of the existing symbolic automatic tools:

AVISPA [ABB+05] provides a common interface for four different back-ends which implement different analysis techniques:

- On-the-fly Model-Checker (OFMC) [BMV05] performs protocol falsification and bounded verification by exploring the transition system of the protocol (over a bounded set of sessions) in a demand-driven way.
- The Constraint-Logic-based Attack Searcher (CL-Atse) [Tur06] applies simplification heuristics and redundancy elimination techniques to analyzes a bounded number of

sessions of the protocol using constraint solving. Note, OFMC and CL-Atse are open to extensions for handling algebraic properties.

– The SAT-based Model-Checker (SATMC) [ACC14] translates the possible finite protocol runs into a propositional formula, which is then given to a SAT solver and any model found is translated back into an attack.

– The Tree Automata based on Automatic Approximation for the Analysis of Security Protocols (TA4SP) [BHKO04] approximates the attacker knowledge by using regular tree languages and rewriting. TA4SP is the only back-end supporting an unbounded number of sessions (for reachability properties) by over-approximation. However, no attack trace is provided by the tool and only the secrecy (reachability) is considered in presence of algebraic properties.

All the four tools take as input a common language called HLPSL (High Level Protocol Specification Language) which is compiled automatically into an intermediate format (named IF). AVISPA was extended in the AVANTSSAR [AAA+12] project, but still relies on the same (although improved) back-ends.

ProVerif [Bla01] is an automatic verification tool based on Horn clauses. It supports trace properties such as secrecy and authentication properties [Bla02], as well as, strong secrecy [Bla04] and equivalences [BAF08] which allows the verification of privacy properties. It can analyze protocols for unbounded number of sessions, and supports user-defined equational theories [AB05a]. ProVerif uses some approximations. It is sound but not complete, and sometimes does not terminate. However, it can reconstructs attack traces if possible [AB05c].

Scyther [Cre08a, Cre08b] verifies protocol using a symbolic backwards search based on ordered patterns. It supports bounded and unbounded number of sessions, however it does not always conclude for the unbounded case. In such a situation however it still gives a verdict for the bounded case. It supports authentication and secrecy properties. It also handles multi-protocols analysis. But no user-definable algebraic functions.

Tamarin [SMCB12, MSCB13] is a security protocol prover supports an unbounded number of sessions. The security properties are specified as multiset rewriting systems with respect to a special subset of (temporal) first-order properties. It also supports Diffie-Hellman exponentiation and a user-defined subterm-convergent rewriting theory.

AKISS [CcCK12] allows to prove trace equivalence properties for bounded processes, featuring user-defined equational theories. It is based on KISS [cCDK12], a tool allowing to prove static equivalence for complex equational theories.

For a comparison of the performance of different tools (in particular Scyther, ProVerif and the different AVISPA back-ends) see [CLN09, LTV09, PBP+10, DSHJ10]. We chose ProVerif for many verification tasks throughout the thesis because of its performance, its support for authentication and equivalence properties (to analyze different notions of Privacy) and its support of user-defined equational theories (to model special cryptographic operations as well as the properties of physical objects (see [DJL14])). Furthermore, it supports the verification of a protocol modeled in a process calculus [AB05a] which resembles the Applied π-Calculus. ProVerif has been used to verify many secrecy, authentication and privacy properties, see *e.g.*, [AB05b, ABF07, KR05, DKR09, BHM08, DRS08, SRKK10, DLL12b, DJP12].

The tools mentioned so far only verify the specification of protocols but not their implementations. Such verification does not tackle side-channel attacks and other implementation attacks, simply because they cannot be modeled without implementing the protocol. Hence, it is necessary to prove security properties on protocols implementations in order to capture the latter attacks. In this context, tools such as [PS10, PSD04, SPP01] have been proposed to translate a model into an implementation using a suitable compiler. So that they can be used to translate a model of formally secure protocol into an implementation, thus reducing the risk of introducing security flaws in the coding phase. Another approach is to extract specification from an implementation in order to formally analyze it. For example, FS2PV [BFGT08] translates a protocol implementation written in a subset of F# into the input language of ProVerif. One can also analyze directly an implementation of a protocol written in a standard programming language such as F#, C or Java. Several tools were adopted and new tools were designed to analyze protocol implementations. The tool CSur [GP05] analyzes protocols written in C by translating them into Horn clauses that can given as an input to H_1 prover [Gou05]. Similarly, JavaSec [Jür06] translates Java programs into first-order logic formulas, which are then given as input to first-order theorem prover e-SETHEO. The tools F7 [BFG10, BBF+11] and F* [SCF+13] use a dependent type system in order to prove security properties of protocols implemented in F#. ASPIER [CD09] uses software model-checking, with predicate abstraction and counter-example guided abstraction refinement in order to verify C implementations of protocols assuming the size of messages and the number of sessions are bounded.

However, such techniques usually suffer from complexity and scalability limitations. For instance, tools that depends on model checking such as ASPIER limits the size of systems that can be verified as model checkers suffer from state explosion problem, and tools based on theorem proving usually involves significant amount of manual effort that, effectively, limits the size of the system that can be verified. Moreover, as formal tools operate on models they may introduce additional proof obligations on the correctness of abstraction or model creation. For example, tools that translate models into implementations require

a proof for the correctness of the used compilers. Note that, the latter tools also suffer from the fact that the protocol modeling language offers less flexibility in the implementation of the protocol than a standard programming language. Furthermore, the problem of finding all possible runtime errors in an arbitrary program is undecidable (going back to Alonzo Church [Chu36] and Alan Turing [Tur 7]).

1.3 Runtime Verification

Another research area concerned with monitoring and analysis of implementations (and systems) executions is runtime verification. It is mainly motivated by the scalability limitations of exhaustive design formal verification. Runtime verification allows to have automatic analysis, that does not require much abstraction, of traces extracted from systems executions. Runtime verification can provide a weaker result than formal verification. Whilst formal verification verifies correctness against all possible system executions, runtime verification only considers traces observed on individual executions. Although incomplete, in the latter sense, runtime verification has an advantage that the actual behavior is analyzed. At the end, runtime verification is not a replacement of formal verification since as famously stated by Edsger Dijkstra decades ago "*testing can only find bugs, not prove their absence*". However, a combination between formal verification and runtime verification can provide an invaluable infrastructure for systems verification which can significantly decrease the complexity of formal verification (for examples see [CK10, CL07, KTAK07, CKK11]). In the thesis, we use runtime verification to monitor executions of exam implementations. Particularly, we analyze real exam executions carried out at *Université Joseph Fourier*.

Runtime verification is a computing analysis based on observing system executions to check certain properties. It performs conventional testing by building monitors from formal specifications. Runtime verification has emerged as a practical application of formal verification. It is particularly useful, when exhaustive design verification is impractical due to inherent complexity of the systems. However, the quality of runtime analysis depends on test scenarios, which makes it less robust compared to formal analysis.

During the last decade, many important tools to monitor systems at runtime have been developed and successfully employed [CR09, MJG+12, CPS09, BH11, RCR15]. To monitor a property is to check whether it is satisfied by a trace, a finite sequence of events, extracted from a system execution. In this context, we can distinguish between the propositional and the parametric approaches. The propositional approach operates on events consisting of simple names without data values. Several tools and techniques adopt the propositional approach. For example, Lee *et al.* [LBaK+98, KVB+99] proposed a Monitoring and Checking (MaC) framework which provides the means to monitor and checks running systems against formally specified requirements. Later, the DMaC [ZSLL09] tool has

been built on the MaC framework and declarative networking (see [LCH+05, LHSR05]) to specify and verify distributed network protocols. Another tool that adopts propositional approach is TemporalRover (TR) [Dru00], which is a commercial runtime verification tool developed by Doron Drusinsky in 2000. TR can be used to verify implementations written in C, C++, Java, Verilog and VHDL, using specifications written in Linear Temporal Logic (LTL) [Pnu77] or Metric Temporal Logic (MTL) [Koy90, AH91]. In the other hand, parametric approaches deal with events that are parameterized by data values. Recently, researchers have focused on parametric monitoring. In parametric approach, we again distinguish between the two following categories:

- *Slicing*: in slicing approach a trace is sliced into a set of propositional values. Slicing technique can be used in combination with any propositional trace analysis by applying the latter on each trace slice. Tools that adopt the slicing approach include: JavaMOP [CR09, MJG+12], Larva [CPS09], and Tracematches [AAC+05, BHL+10]. Recently, Barringer *et al.* introduced Quantified Event Automaton [BFH+12, Reg14], a slicing based formalism for concisely capturing expressive specifications with parameters. They extended the trace slicing approach by allowing free variables, and formalizing the notions of quantification and acceptance. Later, Reger *et al.* [RCR15] developed MarQ, a monitoring Java based tool for QEA.

- *Rule-based*: in this approach a parametric rule system is defined by rules which rewrite a set of facts about the monitored system. Examples of this approach are: the EAGLE [BGHS04] tool, which is based on a restricted first order, fixed-point, linear-time temporal logic with chop over finite traces; the RuleR [BRH10, BHRG09] tool, which is developed as a practically useful and more efficiently executable subset of EAGLE; and the TraceContract [BH11] tool, an API for trace analysis (implemented in the SCALA programming language) which offers writing properties in a notation that combines parameterized state machines with temporal logic.

We use QEA to formalize exam requirements, and we carry out an offline runtime analysis using the MarQ tool. Our choice of using QEA mainly stems from two reasons. First, QEAs is one of the most expressive specification formalism to express monitors. The second reason stems to the fact that QEA is supported by MarQ tool, which came top in the 1st international competition on Runtime Verification [BBF14], showing that MarQ is one of the most efficient existing monitoring tools for both offline and online monitoring [7]. Moreover, QEAs turned out to be a specification language that is accepted and understood by the engineers who were collaborating with us and responsible for the development of the e-exam software.

[7] http://rv2014.imag.fr/monitoring-competition/results

1.4 Contributions and structure of the thesis

In Chapter 2, we first recall the Applied π-Calculus [AF01] which is used throughout the thesis to formally model protocols and properties. Then, we briefly introduce ProVerif [Bla01] tool which is used to carry automatic formal verification. Finally, we provide a brief overview about QEA and MarQ tool which are used to perform runtime verification.

In Chapter 3, we study security properties of exam protocols. Firstly, we propose a formal model for authentication and privacy properties. We apply this model, using ProVerif, to the e-exams due to Huszti & Pethő [HP10] and Giustolisi *et al.* [GLR14], and pencil-and-paper exam at *Université Grenoble Alpes*[8]. Secondly, we propose an abstract model for verifiability properties, and we discuss with the help of ProVerif the verifiability of Giustolisi *et al.* [GLR14] exam, and *Université Grenoble Alpes* exam. Finally, we study exam monitoring at runtime. We propose several monitors expressed as Quantified Event Automata. We validate our monitors by verifying using MarQ real e-exams executions conducted by *Université Joseph Fourier* at pharmacy faculty. Our approach allows to report the individuals responsible of potential failures. Note that, our results shows several flaws on the analyzed exams.

In Chapter 4, we propose a formal framework to define, analyze, and verify security properties of e-cash protocols. We define two client privacy properties and three properties to prevent forgery. Then, we apply our definitions and analyze using ProVerif the Chaum protocol [Cha82], the DigiCash protocol[9], and the Chaum *et al.* protocol [CFN88]. Our analysis confirms several already known attacks, and reveal a flaw confirming the necessity of synchronization in online cash protocols.

In Chapter 5, we model reputation protocols, and formally define several related authentication and privacy properties. Then, we define two verifiability properties in an abstract way. Finally, we validate our model by analyzing, using ProVerif, the security of a simple reputation protocol presented in [PRT04].

In Chapter 6, we consider multiple independent attackers, and show that when analyzing route validity property it is sufficient to verify only five topologies, each containing four nodes, and to consider only three malicious (compromised) nodes. Particularly, we generalize the result obtained in [CDD12] where attackers share their knowledge to the case of attackers that do not share knowledge.

In Chapter 7, we sum up our results and discuss directions for future work.

[8] `www.univ-grenoble-alpes.fr` [9] DigiCash Inc. was an electronic money corporation founded by David Chaum in 1990. The protocol used by DigiCash has been presented by Berry Schoenmakers in [Sch97].

1.5 Publications

All the works presented in this thesis have already been published. The work on authentication and privacy of exam protocols was presented at SECRYPT 2014 [DGK+14]. An extended version was submitted to be published by Springer-Verlag in their "Lecture Notes" (CCIS) series. Note that, this work was accomplished by collaboration with: Jannik Dreier from Institute of Information Security, ETH Zurich; Rosario Giustolisi, Gabriele Lenzini, and Peter Y. A. Ryan from SnT/University of Luxembourg; and Pascal Lafourcade from "Université Grenoble Alpes". The definition of exam verifiability was published at ISPEC 2015 [DGK+15] by collaboration with Jannik Dreier, Rosario Giustolisi, Gabriele Lenzini, and Pascal Lafourcade. The work on runtime verification of e-exam was achieved by collaboration with Yliès Falcone and Pascal Lafourcade from "Université Grenoble Alpes", and was accepted at RV 2015 [KFL15]. The work of Chapter 4 on cash protocols was achieved by collaboration with Jannik Dreier and Pascal Lafourcade, and was presented at SECRYPT 2015 [DKL15][10]. Finally, the results of Chapter 5 on reputation protocols, and Chapter 6 on route validity were achieved by collaborating with Pascal Lafourcade and Yassine Lakhnech from "Université Grenoble Alpes", and were respectively presented at FPS 2014 [KLL14] and FPS 2013 [KLL13].

[10] Our paper on e-cash [DKL15] is selected by ICETE 2015 to submit an extended version in order to be published by Springer-Verlag in their "Lecture Notes" (CCIS) series.

Chapter 2

Preliminaries

In order to formally verify whether a protocol satisfies a certain property, we need first to model them. In this chapter, we introduce the Applied π-Calculus [AF01] in Section 2.1. Then, we present a general overview about ProVerif. Finally, we provide a brief description of the Quantified Event Automata and the MarQ tool in Sections 2.2 and 2.4.

Contents

2.1 Applied π-Calculus

The Applied π-Calculus [AF01] is a process calculus designed for analyzing cryptographic protocols. It is a variant of the π-Calculus [Mil99], a language for modeling and analyzing concurrent systems. The Applied π-Calculus allows a wide variety of cryptographic primitives to be defined using equational theories. Moreover, it is enriched by relations to check equivalences between processes, which allows us to define and analysis privacy properties. In the following, we recall its syntax, semantics and equivalence relations.

2.1.1 Syntax and Semantics

The Applied π-Calculus assumes an infinite set of *names* used to model channels and other atomic data, an infinite set of *variables*, and a *signature* Σ which consists of a finite

set of *function symbols* each with an arity. Function symbols capture primitives used by cryptographic protocols, such as hash functions, digital signatures, encryption and decryption schemes. A function symbol with arity 0 is a constant.

Example 2.1. (**Signature**). *An example of a signature is* $\Sigma_{sym} = \{\texttt{enc}, \texttt{dec}\}$ *which captures symmetric encryption* `enc` *and decryption* `dec`*, each with arity 2. The encryption of message* m *with key* k *is* $\texttt{enc}(m, k)$*, and the decryption of the cipher* c *with the key* k *is* $\texttt{dec}(c, k)$*.*

Terms are defined by names, variables, or function symbols applied to terms according to the grammar shown in Figure 2.1. A term is *ground* when it does not contain variables. We rely on a *type* (or *sort*) system for terms. It includes a set of base types such as

$M, N ::=$	term of type t
n	name of type t
x	variable of type t
$\texttt{f}(M_1, \ldots, M_l)$	application of symbol $\texttt{f} \in \Sigma : t_1 \times \ldots \times t_l \to t$ such that l matches the arity of `f` and M_i is of type t_i

FIGURE 2.1: Grammar for terms

Nonce, **Key** or simply a universal base type **Data**. Additionally, if ω is a type, then **Channel** $\langle\omega\rangle$ is a type (intuitively, the type of a channel transmitting terms of type ω). Names and variables can have any type. Function symbols can only be applied to, and return, terms of base type. Terms equality is modeled using an *equational theory* E, a finite set of equations, which defines an equivalence relation $=_E$ that is closed under substitutions of terms for variables. For simplicity, we may omit the subscript E when it is clear from the context.

Example 2.2. (**Equational Theory**). *The equation* $\texttt{dec}(\texttt{enc}(x, y), y) =_E x$*, where* x *and* y *are variables, says that* `dec`*, representing decryption, is the inverse function of* `enc`*, representing encryption, and that they related in* E *as one expects in a correct symmetric encryption scheme.*

The grammar for *processes* (or *plain processes*) in the Applied π-Calculus is shown in Figure 2.2. The null process 0 does nothing; $P \mid Q$ is the parallel composition of P and Q; and the replication $!P$ executes infinitely many copies of P in parallel. The name restriction $\nu n.P$ generates a new private name n and then continues like P. The conditional process "if $M = N$ then P else Q" behaves as P if $M =_E N$ and as Q otherwise. We may omit sub-term "else Q" when Q is 0. The process $in(u, x).P$ inputs a message on channel u of type **Channel** $\langle\omega_x\rangle$, assigns it to the variable x of type ω_x,

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

FIGURE 2.2: Grammar for plain processes

and then continues like P. Finally, $out(u, M).P$ outputs a term M of type ω_M on the channel u of type **Channel** $\langle \omega_M \rangle$, and then continues like P. The sub-term '$.P$' is usually omitted when P is 0. Plain processes are extended with active substitutions and variable

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

FIGURE 2.3: Grammar for extended processes

restriction. The grammar of *extended processes* is shown in Figure 2.3. The variable restriction $\nu x.A$ bounds the variable x in the process A. The active substitution $\{^M/_x\}$ replaces the variable x with a term M in any process that comes into parallel with it. We use σ to denote a substitution, $x\sigma$ to denote the image of x by σ, and $M\sigma$ to denote the result of applying σ to the term M. Substitutions are well-sorted (*i.e.*, x and $\sigma(x)$ have the same type), and cycle-free (*i.e.*, $dom(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$). Moreover, two active substitutions are not allowed to define the same variable, and there is exactly one defining each restricted variable. We abbreviate tuple of names or variables $u_1, \ldots, u_l$, and tuple of terms $M_1, \ldots, M_l$ to $\tilde{u}$ and $\tilde{M}$, respectively. Accordingly, we abbreviate $\nu u_1, \ldots, \nu u_l.A$ and $\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$ to $\nu\tilde{u}.A$ and $\{^{\tilde{M}}/_{\tilde{x}}\}$, respectively. As an additional notion, we write A^k for $A|\cdots|A$ (k times) where $k \in \mathbb{N}$, in particular $A^0 = 0$ as 0 is the neutral element of parallel composition. Also, following ProVerif, we write *let* $\tilde{x} = \tilde{M}$ *in* P for

$\nu\tilde{x}.(P \mid \{^{\tilde{M}}/_{\tilde{x}}\})$. Note that, names and variables have scopes. A name n is *bound* if it is in the scope of a restriction νn. A variable x is bound if it is in the scope of a restriction νx or of an input $in(u, x)$. Names and variables are *free* if they are not bounded by restrictions or by inputs. We respectively denote by $fv(A)$, $bv(A)$, $fn(A)$ and $bn(A)$ the free variables, bound variables, free names and bound names of the process A. An extended process is closed when all variables are either bound or defined by an active substitution. Bound names and variables can be renamed without changing the semantics of the process, this is called α*-conversion.*

Example 2.3. (α**-conversion**). *Consider the following process, where a, d, m, n, s are names, and x, y are variables, $P_0 = \nu a.\nu n.\nu s.(out(a, n)|in(a, x).out(d, x)|out(d, m)|\{^{s}/_{y}\})$. Then, we have, $fv(P_0) = \{y\}$, $bv(P_0) = \{x\}$, $fn(P_0) = \{d, m\}$, and $bn(P_0) = \{a, n, s\}$. An α-conversion that renames the bound names a, n, and s respectively to c, l, and k, and the bound variable x to z can be applied to P_0 to obtain the following process which has the same semantics as P_0: $\nu c.\nu l.\nu k.(out(c, l)|in(c, z).out(d, z)|out(d, m)|\{^{k}/_{y}\})$.*

A *frame* ϕ is an extended process built up from 0 and active substitutions by parallel composition and restriction. The domain $dom(\phi)$ of a frame ϕ is the set of the variables x for which ϕ contains an active substitution $\{^{M}/_{x}\}$ such that x is not under restriction. Every extended process A can be mapped to a frame $\phi(A)$ by replacing every plain process in A with 0. The domain $dom(A)$ of A is the domain of $\phi(A)$ The frame $\phi(A)$ can be seen as a representation of the static knowledge known about the process to its exterior environment. A context $\mathcal{C}[\_]$ is an extended process with a hole. An evaluation context $\mathcal{C}[\_]$ is a context whose hole is not in scope of replication, a conditional, an input or an output. We say that a context $\mathcal{C}[\_]$ closes A when $\mathcal{C}[A]$ is closed.

Example 2.4. (**Frame**). *Consider again the process P_0 of Example 2.3.*

$$P_0 = \nu a.\nu n.\nu s.(out(a, n)|in(a, x).out(d, x)|out(d, m)|\{^{s}/_{y}\})$$

Then, the frame $\phi(P_0) = \nu a.\nu n.\nu s.(0|0|0|\{^{s}/_{y}\})$, and $dom(A) = dom(\phi(A)) = \{y\}$.

The operational semantics of processes in the Applied π-Calculus is defined by *Structural Equivalence* (see Figure 2.4), *Internal Reduction* (see Figure 2.5), and *Labeled External Transitions* (see Figure 2.6). Two processes with different structure are structurally equivalent if they model the same thing. A process can be structured to a structurally equivalent process using the Structural Equivalence relation ($\equiv$), which is the smallest equivalence relation on extended processes closed under application of evaluation contexts and α-conversion on bound names and bound variables such that the rules of Figure 2.4 hold. The rules for parallel composition and restriction are standard. ALIAS enables the introduction of an arbitrary active substitution with restricted scope.

PAR-0	$A \equiv A \mid 0$	
PAR-A	$A \mid (B \mid C) \equiv (A \mid B) \mid C$	
PAR-C	$A \mid B \equiv B \mid A$	
REPL	$!P \equiv P \mid !P$	
NEW-0	$\nu n.0 \equiv 0$	
NEW-C	$\nu u.\nu v.A \equiv \nu v.\nu u.A$	
NEW-PAR	$A \mid \nu u.B \equiv \nu u.(A \mid B)$	if $u \notin fn(A) \cup fv(A)$
ALIAS	$\nu x.\{^M/_x\} \equiv 0$	
REWRITE	$\{^M/_x\} \equiv \{^N/_x\}$	if $M =_E N$
SUBST	$\{^M/_x\} \mid A \equiv \{^M/_x\} \mid A\{^M/_x\}$	

FIGURE 2.4: Semantics of Structural Equivalence

REWRITE allows to replace a term with an equal term modulo an equational theory. SUBST describes the nature of the active substitution which can be applied to any process that comes in parallel to it. Using structural equivalence, every closed extended process A can be written as $\nu\tilde{n}.\{^{\tilde{M}}/_{\tilde{x}}\} \mid P$, where P is a closed plain process. In particular, every closed frame ϕ can be written as $\nu\tilde{n}.\{^{\tilde{M}}/_{\tilde{x}}\}$.

Example 2.5. (**Structural Equivalence**). *We can use Structural Equivalence to restructure our running process P_0 as follows:*

$$
\begin{aligned}
P_0 &= \nu a.\nu n.\nu s.(out(a,n)|in(a,x).out(d,x)|out(d,m)|\{^s/_y\}) \\
&\equiv \nu a.\nu n.\nu s.(out(a,n)|in(a,x).out(d,x)|\mathbf{0}|out(d,m)|\{^s/_y\}) && \text{PAR-0} \\
&\equiv \nu a.\nu n.\nu s.(out(a,n)|in(a,x).out(d,x)|\big(\nu \boldsymbol{x}.\{^{\boldsymbol{n}}/_{\boldsymbol{x}}\}\big)|out(d,m)|\{^s/_y\}) && \text{ALIAS} \\
&\equiv \nu a.\nu n.\nu s.(\nu \boldsymbol{x}.\big(out(a,n)|in(a,x).out(d,x)|\{^n/_x\}\big)|out(d,m)|\{^s/_y\}) && \text{NEW-PAR} \\
&\equiv \nu a.\nu n.\nu s.(\nu x.\big(\{^{\boldsymbol{n}}/_{\boldsymbol{x}}\}|out(a,n)|in(a,x).out(d,x)\big)|out(d,m)|\{^s/_y\}) && \text{PAR-C} \\
&\equiv \nu a.\nu n.\nu s.(\nu x.\big(\{^n/_x\}|out(a,\boldsymbol{x})|in(a,x).out(d,x)\big)|out(d,m)|\{^s/_y\}) && \text{SUBSET}
\end{aligned}
$$

A process can be evolved internally without any contact with its environment (other processes and the attacker) represented by a context. This can happen based on the Internal Reduction ($\rightarrow$), which is the smallest relation on extended processes closed by Structural Equivalence and application of evaluation contexts such that the rules of Figure 2.5 hold. Note that, we write $P \rightarrow^* P'$ for $P \rightarrow \ldots \rightarrow P'$. The rule COMM allows the internal communication of a variable. This simplicity entails no loss of generality because ALIAS and SUBST can introduce a variable to stand for a term (see Example 2.6). The conditional rules (THEN and ELSE) say that a process takes the P branch if the compared terms are equal under the equational theory $\mathtt{E}$, and that it takes the branch Q otherwise.

COMM	$out(a,x).P \mid in(a,x).Q \to P \mid Q$
THEN	if $M =_E M$ then P else $Q \to P$
ELSE	if $M =_E N$ then P else $Q \to Q$ for any ground terms M, N such that $M \neq_E N$

FIGURE 2.5: Semantics of Internal Reduction

Example 2.6. (**Internal Reduction**). *Consider our running example process P_0 of Example 2.3. Then, we can execute the following transition using Internal Reduction:*

$$\begin{aligned}
P_0 &= \nu a.\nu n.\nu s.(out(a,n)|in(a,x).out(d,x)|out(d,m)|\{^s/_y\}) \\
&\equiv \nu a.\nu n.\nu s.(\nu x.(\{^n/_x\}|out(a,x)|in(a,x).out(d,x))|out(d,m)|\{^s/_y\}) && \text{Example 2.5} \\
&\to \nu a.\nu n.\nu s.(\nu x.(\{^n/_x\}|out(d,x))|out(d,m)|\{^s/_y\}) && \text{COMM} \\
&\equiv \nu a.\nu n.\nu s.(out(d,n))|out(d,m)|\{^s/_y\}) && \text{reverse of Example 2.5}
\end{aligned}$$

An extended process can interacts with its environment based on the Labeled External Transitions rules ($\xrightarrow{\alpha}$) described in Figure 2.6, where α can be an input or an output of a channel name or variable of base type. According to IN, a term M may be input from

IN	$in(a,x).P \xrightarrow{in(a,M)} P\{^M/_x\}$
OUT-ATOM	$out(a,u).P \xrightarrow{out(a,u)} P$
OPEN-ATOM	$\dfrac{A \xrightarrow{out(a,u)} A' \qquad u \neq a}{\nu u.A \xrightarrow{\nu u.out(a,u)} A'}$
SCOPE	$\dfrac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$
PAR	$\dfrac{A \xrightarrow{\alpha} A' \qquad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$
STRUCT	$\dfrac{A \equiv B \qquad B \xrightarrow{\alpha} B' \qquad B' \equiv A'}{A \xrightarrow{\alpha} A'}$

FIGURE 2.6: Semantics of Labeled External Transitions

the external environment. OUT-ATOM allows to output only for free channel names and for free variables of base type. To output restricted channel name, or a term the rule OPEN-ATOM is needed. A term has to be assigned to a variable, which can then be output. This can be done by rewriting $out(a,M).P$ as $\nu x.(out(a,x).P \mid \{^M/_x\})$ using the Structural Equivalence. Note that, the Labeled External Transitions are not closed under evaluation contexts.

Example 2.7. (**Labeled External Transitions**). *Consider our running example process P_0 of Example 2.3. Then, we can execute the following external transition:*

$$
\begin{aligned}
P_0 \quad &= \quad && \nu a.\nu n.\nu s.(out(a,n)|in(a,x).out(d,x)|out(d,m)|\{^s/_y\}) \\
&\equiv && \nu a.\nu n.\nu s.(out(a,x)|in(a,x).out(d,x)|\nu z.(\{^m/_z\}|out(d,z))|\{^s/_y\}) \\
& && \qquad \text{similar to Example 2.5} \\
&\xrightarrow{\nu z.out(d,z))} && \nu a.\nu n.\nu s.(out(a,n)|in(a,x).out(d,x)|\{^m/_z\}|\{^s/_y\}) \\
& && \qquad \textit{by } \text{OUT-ATOM}, \text{ PAR}, \textit{ and } \text{OPEN-ATOM}
\end{aligned}
$$

Additionally, we use the definitions introduced in [DKR09] for the bribed and coerced parties, which are defined by the means of a processes P^{ch_1} and P^{ch_1,ch_2} respectively. The process P^{ch_1} allows us to model parties which are willing to reveal their secret data to the attacker (*e.g.*, a coercer). It is a variant of P that reveals on channel ch_1 (on which the attacker is listening) all its inputs of base type and any freshly generated name of base type. For instance, the process $(\nu n.P)^{ch_1}$ outputs on the channel ch_1 the freshly generated name n if it is of base type, then continues like $P)^{ch_1}$. Similarly, the process $(in(u,x).P)^{ch_1}$ outputs on ch_1 the value it receives if this value is of base type, then continues like P^{ch_1}. However, P^{ch_1} does not forward restricted channel names, as these are used for modeling, *e.g.*, physically secure channels or a public key infrastructure which securely distributes keys. Note that in the latter example the keys are forwarded to the attacker, but not the secret channel names on which the key are received.

Definition 2.1. (**Process P^{ch_1} [DKR09]**). *Let P be a plain process and ch_1 be channel name. The process P^{ch_1} is defined follows:*

- $0^{ch_1} \mathrel{\hat{=}} 0$,
- $(P|Q)^{ch_1} \mathrel{\hat{=}} P^{ch_1}|Q^{ch_1}$,
- $(!P)^{ch_1} \mathrel{\hat{=}} !P^{ch_1}$,
- $(\nu n.P)^{ch_1} \mathrel{\hat{=}} \nu n.out(ch_1,n).P^{ch_1}$ *if n is a name of base type, or* $(\nu n.P)^{ch_1} \mathrel{\hat{=}} \nu n.P^{ch_1}$ *otherwise,*
- $(\textit{if } M =_E N \textit{ then } P \textit{ else } Q)^{ch_1} \mathrel{\hat{=}} \textit{if } M =_E N \textit{ then } P^{ch_1} \textit{ else } Q^{ch_1}$,
- $(in(u,x).P)^{ch_1} \mathrel{\hat{=}} in(u,x).out(ch_1,x).P^{ch_1}$ *if x is a variable of base type, or* $(in(u,x).P)^{ch_1} \mathrel{\hat{=}} in(u,x).P^{ch_1}$ *otherwise,*
- $(out(u,x).P)^{ch_1} \mathrel{\hat{=}} out(u,x).P^{ch_1}$.

The application of the transformation is distributed on the parallel composition '|' and on the replication '!' since in these two cases it is enough for each parallel process to output its data that are required to reveal. In the case of the conditional process, the

transformation is applied to the taken branch. Finally, $0^{ch_1} \hat{=} 0$ as the null process does nothing, and $(out(u,x).P)^{ch_1} \hat{=} out(u,x).P^{ch_1}$ as the output messages are not revealed. Note that in the remainder, we assume that $ch_1 \notin fn(P) \cup bn(P)$ before applying the transformation.

The second process P^{ch_1,ch_2} does not only reveal the secret data on channel ch_1, but also takes orders from an outsider (attacker) on the channel ch_2 before sending a message or branching. This models a completely corrupted party. For instance, the attacker can control which branch to take in the case of the conditional process, and can provide a message of his choice to output in the case the output process.

Definition 2.2. (Process P^{ch_1,ch_2} [DKR09]). *Let P be a plain process and ch_1, ch_2 be two channel names. The process P^{ch_1,ch_2} is defined follows:*

- $0^{ch_1,ch_2} \hat{=} 0$;
- $(P|Q)^{ch_1,ch_2} \hat{=} P^{ch_1,ch_2}|Q^{ch_1,ch_2}$;
- $(!P)^{ch_1,ch_2} \hat{=} !P^{ch_1,ch_2}$;
- $(\nu n.P)^{ch_1,ch_2} \hat{=} \nu n.out(ch_1,n).P^{ch_1,ch_2}$ *if n is a name of base type, otherwise* $(\nu n.P)^{ch_1,ch_2} \hat{=} \nu n.P^{ch_1,ch_2}$;
- $(\textit{if } M =_E N \textit{ then } P \textit{ else } Q)^{ch_1,ch_2} \hat{=} in(ch_2,x).\textit{if } x = \textit{true then } P^{ch_1,ch_2} \textit{ else } Q^{ch_1,ch_2}$, *where x is a fresh variable and true is a constant;*
- $(in(u,x).P)^{ch_1,ch_2} \hat{=} in(u,x).out(ch_1,x).P^{ch_1,ch_2}$ *if x is a variable of base type, otherwise* $(in(u,x).P)^{ch_1,ch_2} \hat{=} in(u,x).P^{ch_1,ch_2}$;
- $(out(u,x).P)^{ch_1,ch_2} \hat{=} in(ch_2,x).out(u,x).P^{ch_1,ch_2}$, *where x is a fresh variable.*

To hide the outputs of an extended process on a certain channel, we use the following definition.

Definition 2.3. (Process $A^{\backslash out(ch,\cdot)}$ [DKR09]). *Let A be an extended process and ch be a channel name. We define the process $A^{\backslash out(ch,\cdot)}$ as $\nu ch.(A \mid !in(ch,x))$.*

The process $A^{\backslash out(ch,\cdot)}$ is as the process A, but hiding the outputs on the channel ch. We also use the following two lemmas from [DKR09].

Lemma 2.1. (Process $(P^{ch})^{\backslash out(ch,\cdot)}$ [DKR09]). *Let P be a closed plain process and ch a channel name such that $ch \notin fn(P) \cup bn(P)$. Then, we have that $(P^{ch})^{\backslash out(ch,\cdot)} \approx_l P$.*

Lemma 2.2. (Context Commutativity [DKR09]). *Let $\mathcal{C}_1 = \nu\tilde{u}_1.(\_|P_1)$ and $\mathcal{C}_2 = \nu\tilde{u}_2.(\_|P_2)$ be two evaluation contexts such that $\tilde{u}_1 \cap (fv(P_2) \cup fn(P_2)) = \emptyset$, and $\tilde{u}_2 \cap (fv(P_1) \cup fn(P_1)) = \emptyset$. Then, $\mathcal{C}_1[\mathcal{C}_2[A]] \equiv \mathcal{C}_2[\mathcal{C}_1[A]]$ for any extended process A.*

2.1.2 Equivalences

Two processes are equivalent if an external observer cannot tell them apart. The Applied π-Calculus has two relations to define equivalence between processes are *Observational Equivalence*, and *Labeled Bisimilarity*. Two processes are observationally equivalent if for every context each output or internal transition of the first process can be simulated by the second process.

Definition 2.4. (Observational Equivalence [AF01]). Observational Equivalence ($\approx$) *is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A\ \mathcal{R}\ B$ implies:*

1. *if $A \Downarrow a$, then $B \Downarrow a$, where $A \Downarrow a$ means that the process A sends a message on the channel a, i.e., when $A \rightarrow^* \mathcal{C}[out(a, M).P]$ for some evaluation context $\mathcal{C}[\_]$ that does not restrict a;*
2. *if $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B';*
3. *$\mathcal{C}[A]\ \mathcal{R}\ \mathcal{C}[B]$ for all closing evaluation contexts $\mathcal{C}[\_]$.*

Example 2.8. (**Observational Equivalence**). *Let* `f` *be a unary function symbol with no equations. Then, we have $\nu s.out(a, s) \approx \nu s.out(a, h(s))$, although this is not easy to prove. As an another example, we have that $out(a, s_1) \not\approx out(a, s_2)$ where s_1 and s_2 are names. We can prove that using the context $\mathcal{C}[\_] \equiv in(a, x).$ if $x = s_1$ then $out(a, s_1)|\_$, which does not satisfies the clause 3 of Definition 2.4. Indeed, we have that $\mathcal{C}[out(a, s_1)] \rightarrow^* out(a, s_1)$ thus $\mathcal{C}[out(a, s_1)] \Downarrow a$, but $\mathcal{C}[out(a, s_2)]$ can never outputs on a.*

It is difficult to prove Observational Equivalence due to the quantification over all contexts. Therefore, Labeled Bisimilarity ($\approx_l$) is introduced, which is more suitable for both manual and automatic reasoning. Labeled Bisimilarity relies on the notion of *Static Equivalence* ($\approx_s$), which is based on the equivalence of two terms in a given frame. Note that, the two relations $\approx$ and $\approx_l$ coincide [AF01, Liu11], that is, for any two closed extended processes A and B we have $A \approx B$, if and only if, $A \approx_l B$.

Definition 2.5. (Equality in a Frame [AF01]). *Two terms M and N are* equal in the frame ϕ, *written $(M = N)\phi$, if and only if, $\phi \equiv \nu\tilde{n}.\sigma$, $M\sigma = N\sigma$ and $\{\tilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$, for some names $\tilde{n}$ and substitution σ.*

Definition 2.6. (Statical Equivalence [AF01]). *Two closed frames ϕ and ψ are* statically equivalent, *written $\phi \approx_s \psi$, if $dom(\phi) = dom(\psi)$, and if for all terms M and N, we have that $(M = N)\phi$, if and only if, $(M = N)\psi$. Two extended processes A and B are statically equivalent, written $A \approx_s B$ if their frames are statically equivalent.*

Two frames are statically equivalent if any test ($M = N$) that holds on one frame should also hold on the other frame. Two processes are statically equivalent if all their previous operations gave the same results, so that they cannot be distinguished from the messages they previously exchanged with the environment. However, they may evolve in different ways in future.

Example 2.9. (**Statical Equivalence [CK14]**). *Let* $\phi_1 = \nu k.\{^{k}/_{x}, ^{\mathtt{enc}(0,k)}/_{y}\}$ *and* $\phi_2 = \nu k.\{^{k}/_{x}, ^{\mathtt{enc}(1,k)}/_{y}\}$. *Then,* $\phi_1 \not\approx_s \phi_2$ *since we have that* $(\mathtt{enc}(0,x) = y)\phi_1$, *but* $(\mathtt{enc}(0,x) \neq y)\phi_2$. *Adding some randomness to the encryption will result into statically equivalent frames. For instance,* $\nu k.\nu r.\{^{k}/_{x}, ^{\mathtt{enc}((0,r),k)}/_{y}\} \approx_s \nu k.\nu r.\{^{k}/_{x}, ^{\mathtt{enc}((1,r),k)}/_{y}\}$.

The idea of Static Equivalence can be extended to *labeled bisimilarity.*

Definition 2.7. (Labeled Bisimilarity [AF01]). Labeled bisimilarity ($\approx_l$) *is the largest symmetric relation* $\mathcal{R}$ *on closed active 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'.

Example 2.10. (**Labeled Bisimilarity [RS11]**). *For any closed process* P, *we have*

$$\nu a.(out(a,m).P \mid in(a,x).Q) \approx_l \nu a.(P \mid Q\{^{m}/_{x}\})$$

The equivalence holds because the only choice that the left-side process can take, and which the right-side process cannot is the internal reduction over the private channel a *by COMM. This reduction results in the right-side process.*

2.2 ProVerif Tool

Throughout the thesis, we use ProVerif to perform automatic verification tasks. ProVerif is an automatic verification tool based on Horn clauses originally developed by Bruno Blanchet [Bla01] to verify secrecy (reachability). It is then extended to support correspondence properties between events to verify authentication properties [Bla02], strong secrecy [Bla04], and equivalences [BAF08] which allows the verification of privacy properties. It also supports user-defined equational theories [AB05a], which allows the modeling of a large class of cryptographic primitives, as well as, the properties of physical objects. Furthermore, Küster and Truderung have shown how to reduce the derivation problem for Horn theories with XOR ($\oplus$) to the XOR-free case in [KT11]. Their reduction allows one to carry out the analysis of the protocols that involve XOR operator using tools,

such as ProVerif. The authors then adopted their approach to the case of Diffie-Hellman exponentiation [KT09].

ProVerif takes a protocol and a property modeled in a process calculus [AB05a] which resembles the Applied π-Calculus, and translates them into Horn clauses (Prolog rules). ProVerif then determines whether the property is satisfied by the protocol or not. In case of failure, ProVerif may provide a trace of the obtained attack [AB05c]. For more details about modeling protocols in ProVerif and its output we refer to the manual [BSC15]. Note that, ProVerif assumes a Dolev-Yao like attacker that can intercept all messages, compute new messages from the messages it has received, and send any message it can build.

Secrecy of a term (can the attacker get the secret s?) is defined in ProVerif as the attacker cannot obtain the term by communicating with the protocol and performing computations. It is modeled as a predicate: `query attacker`(s). ProVerif determines whether the term s can be inferred from the Prolog rules. Authentication is captured by correspondence assertions of events "on every execution trace an event $\mathtt{e_2}$ is preceded by an event $\mathtt{e_1}$". Events are annotations that do not change the behavior of the processes, but help to reason about authentication properties. A basic correspondence assertion is a formula of the form: $\mathtt{e_2}(N) \Longrightarrow \mathtt{e_1}(M)$. That is, for each occurrence of the event $\mathtt{e_2}(N)$ there is a previous execution of event $\mathtt{e_1}(M)$. One can also define some relations (to be satisfied) between M and N. A more general correspondence can be defined by replacing the event $\mathtt{e_1}(M)$ by conjunctions and disjunctions of events. ProVerif can also capture the one-to-one relationship between events using injective correspondence, that is: for each occurrence of event $\mathtt{e_2}(N)$ there is a distinct earlier execution of event $\mathtt{e_1}(M)$. Moreover, ProVerif supports nested correspondence, that is some of the events in the right hand side of $\Longrightarrow$ are replaced with correspondences.

To capture strong secrecy and other privacy properties ProVerif provides the ability to verify observational equivalences. Strong secrecy of a value x can be questioned using `noninterf` x. Other privacy properties such as anonymity are questioned using an equivalence between two processes that differ only in the choice of some terms. Such an equivalence is written in ProVerif by a single *biprocess* that encodes the two processes by using the construct `choice`$[M, N]$ to represent the terms that differ between the two processes, where the first component of the choice M is used in the first process while the second component N is used in the second process. ProVerif can also prove equivalence $P \approx Q$ between two processes P and Q presented separately, using the command `equivalence` P Q where P and Q are processes that do not contain `choice`. ProVerif will in fact try to merge the processes P and Q into a biprocess and then prove equivalence of this biprocess. Note that ProVerif is not always capable of merging two processes into a biprocess: the structure of the two processes must be fairly similar.

As we noted in the Chapter 1 ProVerif is sound but not complete. This means that, if ProVerif claims that a property is true or false, then this claim is correct. However, ProVerif may say that a property "cannot be proved", which is a "do not know" answer. Furthermore, ProVerif may not terminate. The incompleteness is due to some approximations that are performed by the translation of protocol into Horn clauses. The main abstractions are (more details can be found in [BSC15]):

- Actions can be repeated any number of times.
- Generation of a fresh name n is represented as functions of the inputs located above νn. So the more the νn is moved downward in the process, the more arguments they have, and in general the more precise (but more costly) the analysis is.
- Some approximations are made when dealing with private channels. Particularly, when a is a private channel, the process P in $out(a, M).P$ can only be executed when some input listens on channel a; ProVerif does not take that into account and considers that P can always be executed.
- Further approximations are made when proving observational equivalence. In order to show that P and Q are observationally equivalent, ProVerif proves that at each step P and Q reduce in the same way: the same branch of a test or destructor application is taken, communications happen in both processes or in neither of them. This property is sufficient for proving observational equivalence, but it is not necessary. For instance, the biprocess $out(a, \texttt{choice}[m, n]) \mid out(a, \texttt{choice}[n, m])$ satisfies observational equivalence as the difference in the first output is compensated by the second output, but ProVerif cannot show this. For ProVerif to prove observational equivalence, we have to rewrite the biprocess into the structurally equivalent one $out(a, \texttt{choice}[m, m]) \mid out(a, \texttt{choice}[n, n])$. It becomes more difficult when a configuration similar to the one above happens in the middle of the execution of the process. We faced such a problem when we analyze some of our case studies with ProVerif. We tackle it by defining an additional process that takes the outputs messages on private channels, shuffles them, and then outputs them on a public channel. Note that, Ben Smyth *et al.* are working on an extension of ProVerif to tackle such cases [DRS08].

2.3 Quantified Event Automata

In this section, we present the Quantified Event Automata, an expressive formalism to represent parametric specifications. We refer to [BFH+12, Reg14] for full syntax and definitions.

A finite-state automaton is a mathematical model of the allowable operations used to design, *e.g.*, computer programs. It is conceived as an abstract machine that can be in one of a finite number of states. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. An event automaton (EA) is a non-deterministic finite-state automaton whose alphabet consists of parametric events and whose transitions may be labeled with guards and assignments. We use the notation $\frac{[guard]}{assignment}$ to write guards and assignments on transitions: ($:\hat{=}$) for variable declaration then assignment, ($:=$) for assignment, and ($=$) for equality test. A quantified event automaton (QEA) is a generalization of event automaton where zero or more of the EA variables are quantified. It formally defines a language (*i.e.*, a set of accepted traces) over instantiated parametric events. Acceptance is decided by replacing these quantified variables by each value in their domain to generate a set of EA and then using the quantifiers to determine which of these EA must accept the given trace. Unquantified variables are left free, and they can be manipulated through assignments and updated during the processing of the trace. Moreover, new free variables can be introduced while processing the trace.

To illustrate Quantified Event Automata and their languages, we consider the following example: "for any i: $\mathtt{e}_2(i)$ is preceded by an event $\mathtt{e}_1(i)$" (see Figure 2.7). The associated input alphabet contains only the events $\mathtt{e}_1(i)$ and $\mathtt{e}_2(i)$, so any other events in the trace are ignored. The QEA of Figure 2.7 has two final (accepting) states (shaded in gray),

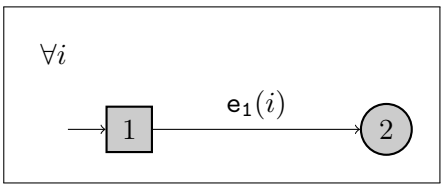


FIGURE 2.7: A QEA describing $e_2(i)$ is preceded by an event $e_1(i)$.

zero failure states (white color is used to represent failure states), one quantified variable `i`, and zero free variables. The initial state has an arrow pointing to it. Square states are closed to failure (next) states, *i.e.*, if no transition can be taken, then there is a transition to an implicit failure state. Whereas circular states are closed to self (skip) states, *i.e.*, if no transition can be taken, then there is an implicit self-looping transition. The empty trace is accepted by the QEA since the initial state is accepting. State (1) is a square state, so an event $\mathtt{e}_2(i)$ that is not preceded by event $\mathtt{e}_1(i)$ leads to a failure. An event $\mathtt{e}_1(i)$ in state (1) leads to state (2) which is a skipping state, so after event $\mathtt{e}_1(i)$ any sequence of events (for same i) is accepted. The quantification $\forall i$ means that the property must hold for all values that `i` takes in the trace, *i.e.*, the values obtained when matching the symbolic events in the specification with concrete events in the trace. For instance, consider the following trace: $\mathtt{e}_1(i_1).\mathtt{e}_2(i_2).\mathtt{e}_2(i_1).\mathtt{e}_1(i_2)$. To decide whether it is accepted

or not, the trace is sliced based on the values that can match i, resulting in the two slices: $i \mapsto i_1$: $\mathtt{e_1}(i_1).\mathtt{e_2}(i_1)$, and $i \mapsto i_2$: $\mathtt{e_2}(i_2).\mathtt{e_1}(i_2)$. Then, each slice is checked against the event automaton instantiated with the appropriate value for i. The slice associated to i_1 is accepted as it reaches the final state (2), while the slice associated to i_2 does not reach a final state since event $\mathtt{e_2}(i_2)$ leads from state (1) to an implicit failure state. Therefore, the whole trace is not accepted by the QEA.

2.4 MarQ Tool

The MarQ [RCR15] tool is suitable for offline (on reduced system executions) and online (on running system) monitoring of Java programs using AspectJ [KHH+01] for instrumentation. It won the offline monitoring and online monitoring for Java tracks in the 1st international Runtime Verification competition [BBF14]. Typically runtime monitoring consists of three stages: firstly, a property denoting a set of valid traces is specified in a formal language. Secondly, the system of interest is instrumented to produce the required events recording information about the state of the system. Thirdly, a monitor is generated from the specification, which processes the trace to produce a verdict.

To specify QEA as input to the MarQ system, QEABuilder is used. A builder object is used to add transitions and record information about quantification and states. These are used to construct a QEA that is passed to the MonitorFactory. Figure 2.8 shows how QEABuilder can be used to construct the QEA given in Figure 2.7.

```
QEABuilder q = new QEABuilder("Correspondence Assertion");
int E1 = 1; int E2 = 2;
int i=-1;
q.addQuantification(FORALL, i);
q.addTransition(1,E1, new int[]{i},2);
q.addFinalStates(1,2);
q.setSkipStates(2);
```

FIGURE 2.8: Using QEABuilder to construct the QEA of Figure 2.7.

A builder object `q` is first created. Then, event names and variables are declared. Quantified variables are negative integers, whereas free variables are positive integers. After that, the universal quantifications are identified. In our example we have one quantified variable `i`. Transitions are then added using `addTransition`. For each transition, the start state, event name, parameters, guards, assignments, and end state are specified. The builder object of Figure 2.7 has one transition from state (1) to state (2) labeled with event `E1` which has one parameters `i`. This single transition has neither

guards nor assignments. However, MarQ tool includes a library of guards and assignments for dealing with equality, integer arithmetic, and sets. Moreover, it is possible for the user to define a new guard or assignments by implementing a simple interface. The final step of QEA building is to specify the accepting states, and skipping states. By default, states are failure and next states. Once the QEA is constructed the monitor can be created by a call to the `MonitorFactory`. This will inspect the structure of the QEA and produce an optimized monitor. Two modes of monitoring are possible: offline monitoring and online monitoring. In offline monitoring, a trace in CVS or XML format is given as a log file and processed by a translator then by the monitor to produce a verdict. The translator converts each event from "string" into the format used by the monitor. Whereas in online monitoring, a program is instrumented to emit events which are feed to the monitor. The monitor then produces a verdict on each event. Note that, in case of online monitoring it is necessary to deliver each event to the monitor at the time it is generated by the system. This can be done, for example, using AspectJ [KHH+01]. Details about monitoring process and the algorithm used in MarQ can be found in [Reg14].

2.5 Summary

In this chapter, we recalled the Applied π-Calculus [AF01], the formal language we use in the thesis to model protocols. Then, we provided an overview about ProVerif, the tool we use to perform symbolic automatic verification tasks throughout the thesis. Finally, we given a brief description of the Quantified Event Automata and the MarQ tool, as we use them in Chapter 3 respectively to model exam requirements and to perform runtime monitoring for real exam executions.

Chapter 3

Exam Protocols

Exams are systems employed to assess the skills, or the knowledge of candidates. In an exam process, candidates may cheat to get a higher mark, examiners may be bribed, and exam authorities may manipulate the results. Thus, security is significant to ensure fairness and correctness.

In this chapter, we propose a formal model for authentication and privacy properties of exams. We analyze using ProVerif the exams due to Huszti & Pethő [HP10], Giustolisi *et al.* [GLR14], and pencil-and-paper exam at *Université Grenoble Alpes*[1] (in short, Grenoble exam). Then, we propose an abstract model of verifiability, and discuss with the help of ProVerif the verifiability of Giustolisi *et al.* [GLR14] exam, and Grenoble exam. Finally, we study exam monitoring at runtime. We propose several monitors, expressed as Quantified Event Automata, to monitor the main properties of e-exams. We validate our monitors by verifying, using the Java tool MarQ, real e-exam executions conducted by *Université Joseph Fourier*[2] at pharmacy faculty. Our approach allows to report the individuals responsible of potential failures.

Contents

[1] www.univ-grenoble-alpes.fr [2] www.ujf-grenoble.fr

3.1 Introduction

Exams are assessment tools used to measure skills, or knowledge. Typically, candidates sit for the exam, and submit some answers which are collected by an authority; examiners evaluate the candidates' answers and deliver marks; and finally marks are notified by an authority to the candidates.

Traditionally, exams are taken pencil-and-paper at hand. In contrast, nowadays several exams rely on information and communication technology. They are called *electronic exams*, in short e-exams. For instance, universities such as MIT, Stanford, and Berkeley, just to cite a few, have began to offer university courses remotely using the Massive Open Online Course platforms (*e.g.*, Coursera[3] and edX[4]) which offer e-exams. Even in a less ambitious and more traditional setting, universities start adopting e-exams to replace traditional exams, especially in the case of multiple-choice questions and short open answers. For example, pharmacy exams at *Université Joseph Fourier* (UJF) have been organized electronically using tablet computers since 2014 [Fig15]. Other institutions, such as ETS[5], CISCO[6], and Microsoft[7], have for long already adopted their own platforms to run, generally in qualified centers, electronic tests required to obtain their program certificates.

To ensure fairness between candidates, it is important to prevent cheating, and manipulation of answers and marks during the exam process. In general, candidates may try to cheat in order to get a higher mark. Candidates' cheating is the main concern of exam authorities. Usually, authorities try to prevent the candidates from cheating with invigilated tests, and by using student cards, or login/password to authenticate the candidates. Even for remote exams where it is not possible to have human invigilators (proctors), a software running on the student computer is used, *e.g.*, ProctorU[8], and webcam or biometrics may be used for authentication. However, such measures are insufficient, as the trustworthiness and the reliability of exams are today threatened not only by candidates. Indeed, threats and errors may come from the use of information technology, as well as, from bribed examiners and dishonest exam authorities which are willing to tamper with exams as recent scandals have shown. For example, in the

[3] www.coursera.org [4] www.edx.org [5] www.etsglobal.org [6] www.cisco.com
[7] www.microsoft.com/learning/en-us/default.aspx [8] www.proctoru.com

Atlanta scandal, school authorities colluded in changing student marks to improve their institution's rankings and get more public funds [Lar13]. All such diverse exam systems must be verified for the presence/absence of threats and irregularities. However, properties of exam protocols have to be clearly defined and formalized before. In this chapter, we formalize and analyze the following security properties of exam protocols:

- **Authentication:** Exam authority concerned in *Answer Origin Authentication* which ensures that only registered candidates take the exam, and that one exam-form (copy) is collected from each candidate. For the candidate it is crucial to preserve the associations between her identity, her answer, and her mark. We define three properties for that purpose, each covers a certain exam phase: i) *Form Authorship* ensures that the contents of every accepted (collected) exam-form (identity, questions, and answers) are not modified after submission, ii) *Form Authenticity* ensures that the content of every exam-form is not modified after the collection and until after the form is marked by an examiner, iii) *Mark Authenticity* ensures that every candidate receives the mark which was assigned by the examiner to his exam-form.

- **Privacy:** During examination process the anonymity of critical parties has to be ensured in order to prevent bribing, and guarantee fairness among the candidates to avoid favoritism. For instance, in order to guarantee the main goal of the exam, which is assessing the knowledge and skills, the exam questions should not be disclosed until the examination phase begins, this is ensured by our first privacy property *Question Indistinguishability*. To prevent favoritism between candidates at marking, an exam protocol should also satisfies privacy notions like, *Anonymous Marking*, which ensures that the examiner evaluates the exam-forms without being aware of their authors, and *Anonymous Examiner* which prohibits a candidate from knowing which examiner is going to grade his exam-form, this is to avoid bribing. Finally, to preserve the particularity of each candidate some notions are needed like *Mark Privacy* which ensures that a candidate's mark remains secret, and *Mark Anonymity*, a weaker variant, which ensures that the attacker is unable to associate a mark to its corresponding candidate.

- **Verifiability:** A verifiable exam protocol should allow a candidate to verify that her questions are valid, that her answers were not manipulated, and that her mark is correct. It should also allow a generic observer (*e.g.*, judge) to verify that only registered candidates participated in the exam, that no answer were manipulated, that all answers were marked correctly, and that all marks were assigned for corresponding candidates.

– **Monitoring:** The verifiability properties operate on abstract models of protocol specification. However, some errors may introduced by the implementation. In order to ensure that protocol executions are correct, we discuss exam monitoring. We propose several monitors based on Quantified Event Automata, which allows to verify actual exam executions at runtime. Namely, no unregistered candidate try to participate in the exam by submitting an answer; answers are accepted only from registered candidates; all accepted answers are submitted by candidates, and for each question at most one answer is accepted per candidate; all candidates answer the questions in the required order; answers are accepted only during the examination time; another variant of the latter that offers flexibility in the beginning and the duration of the exam; all answers are marked correctly; and the correct mark is assigned to each candidate. Our formalization also allows us to detect the cause of the potential failures and the responsible parties.

Note that, our definitions apply not only to educational assessments and skill tests, but also to situations where the properties discussed above are desirable such as peer review systems, project proposal, public tender, and benchmarks. For example in academic peer reviews, we have authors which submit academic papers in analogs to candidates which submit answers in educational assessments. The papers are then evaluated by some reviewers (in analogue to examiners) after being collected by a certain authority. Note that, properties such as *Form Authorship*, *Form Authenticity*, and *Anonymous Marking* are highly desirable in peer review systems.

Contributions. In this chapter, we provide the following contributions:

– In the first part, we propose a formal framework in the Applied π-Calculus to model and analyze the authentication and privacy properties of exam protocols. We validate the proposed framework by analyzing using ProVerif three case studies: the protocols by Huszti & Pethő [HP10], Giustolisi *et al.* [GLR14], and Grenoble exam.

– In the second part, we propose abstract definitions of verifiability properties related for exam protocols. We apply the proposed verifiability definitions to the protocol by Giustolisi *et al.*, and Grenoble exam, and again we use ProVerif to verify these two protocols.

– In the final part, we define several QEAs which allow to monitor at runtime exam requirements of exams. We also define for each property an alternative variant that additionally collects and reports at the end some data in case of failure. Then, we implement these QEAs using MarQ tool, and we perform offline monitoring, based on the available data logs, for an e-exam organized by *Université Joseph Fourier*.

Outline of the Chapter. In Section 3.2, we discuss the related work. We formally model exam protocol in Applied π-Calculus to define authentication and privacy properties in Section 3.3.1. Then, we define the authentication and privacy properties in Section 3.3.2. In Section 3.3.3, we analysis the authentication and privacy properties of our three case studies. In the second part (Section 3.4): we define an abstract model of exam protocols and our verifiability properties. We split our properties into individual (Section 3.4.1) and universal properties (Section 3.4.2). Then, we analysis the two related case studies in Section 3.4.3. In the final part (Section 3.5): we introduce QEA syntax and present the events we considered to define the monitors in Section 3.5.1. We defined our monitors as QEAs in Section 3.5.2. Then, we perform an analyze of actual e-exam executions at UJF in Section 3.5.3. We finally conclude in Section 3.6.

3.2 Related Work

In this section we discuss related work on exam protocols, and the link to other applications such as auction and voting.

Exam Protocols. The majority of works on exam protocols argue the security of the proposed protocols only informally. Castellà-Roca *et al.* [CHD06] proposed an e-exam system, and claimed that it guarantees a number of authentication and privacy properties in presence of a trusted exam manager. Huszti & Pethő [HP10] proposed an exam protocol which relies on the *reusable anonymous return channel* [GJ03]. The proposed protocol considers minimal trust requirements, but a trusted registry. Giustolisi *et al.* [GLB13] listed some relevant requirements for exam protocols, yet only informally. Latter, an extension for these requirements and an internet-based exam protocol, Remark! protocol, have been presented in [GLB13]. The authors also provided some arguments to show that their protocols meets the presented requirements. The majority of these requirements are formalized herein in this thesis. Bella *et al.* [BGL14] presented an exam protocol (WATA IV), which relies on visual cryptography [NS94]. It considers corrupted examiners, but assumes a *honest-but-curious* anonymizer. WATA IV is the latest in a family of protocols [BCR10, BCCR11, BGL14] with prototypes used to run exams at University of Catania [BCCR11]. WATA IV was informally analyzed by its authors who claim that WATA IV meets several authentication and privacy properties.

Few other works formally tackle exam protocols. Foley *et al.* [FJ95] proposed a formalization for functionality and confidentiality requirements of Computer Supported Collaborative Working (CSCW) applications. They illustrated their formalization using an exam example as a case study. Arapinis *et al.* [ABR13] propose a cloud-based protocol for conference management system that supports applications, evaluations, and decisions. They identify a few privacy properties (secrecy and unlinkability) that should hold despite

a *malicious-but-cautious* cloud, and they prove, using ProVerif, that their protocol satisfies them. Recently in 2015, Bella *et al.* [BGLR15] presented an exam protocol that does not require trusted third party (TTP). They also analyzed the proposed protocol using Proverif. The considered six out of the nine authentication and privacy properties we formalize in this thesis. All our exam privacy properties but *Anonymous Examiner* are considered. Concerning authentication properties, they consider *Answer Origin Authentication* and *Form Authorship*. They also proposed a new definition for *Mark Authenticity*, which says that "an examiner correctly attributes the mark computed on a given answer to the candidate that submitted this answer". Whereas, our definition of *Mark Authenticity* says that a candidate receives the mark attributed by an examiner to the answer handed by the exam authority as it is accepted from this candidate. Note that, our property *Form Authenticity* guarantees that the examiner computes the mark on the answer accepted from the candidate (that is the correct answer is handed by the authority to the examiner). So together our two properties *Mark Authenticity* and *Form Authenticity* ensure that a candidate receives the mark computed on the answer accepted from him. This is a bit stronger than their definition of *Mark Authenticity* which only guarantees that a mark is attributed to a candidate, but not necessarily received by him. Note also that, the authors of the paper [BGLR15] used a different meaning for the event `notified`. That is, it emitted when an examiner attributes a given mark to a given candidate. Whereas, in our definition `notified` is emitted when the candidate receives the mark. Additionally, they defined the following four properties:

- *Candidate Authorisation* which says that only registered candidates can take the exam. Note that, we consider a similar property (*Candidate Registration*) to monitor exam executions. Our goal is to check if an unregistered candidate was trying to spoof the system by submitting an answer. However, when one wants to check protocol specification, we think what is important is to be sure that no answer will be accepted from an unregistered candidate (which can submit answers). The latter is ensured by our property *Answer Origin Authentication*.

- *Notification Request Authentication* which says that a mark can be associated with the candidate only if she requests to learn her mark. The intuition behind this unusual property is to allow a candidate to withdraw from the exam before notification.

- *Mark Verifiability* which says that a candidate can verify that she has been assigned with the mark attributed to her answer. This property is similar to our property *Correct Mark Reception*. Additionally, we propose a property *Mark Correctness* that allows a candidate to check that her mark was computed correctly. Moreover, we propose several properties that allow a candidate, as well as, an external observer to verify several steps during the examination process (see Section 3.4).

– An accountability property, *Testing Dispute Resolution*, which allows to identify the responsible party when a candidate fails to submit an answer or to receive the corresponding mark. Note that, we do not tackle accountability in this thesis. However, each of our verifiability properties covers a certain step in the examination process which helps to identify the source of potential failures. Moreover, our monitors for exam executions allow us to identify the responsible parties in case of failure.

The analysis shows that the proposed protocol satisfies all the considered properties.

Link to the Other Applications. There are several works presenting the formalization and verification of authentication and privacy properties in domains that seem related to e-exams, namely e-voting [DLL11, DLL12b, DLL12a, BHM08, DKR09, DKR06] and e-auction systems [DJP10, DLL13, DJL13]. Some of the security properties therein studied remind those we are presenting for e-exams. For instance, *Answer Origin Authentication* is analogous to voter and bidder authentication. *Mark Privacy* reminds ballot privacy and losing bids privacy. Yet, there are fundamental differences. In e-exams, *Answer Authorship* should be preserved even in the presence of colluding candidates. Conversely, vote (bid) authorship is not a problem for e-voting (e-auction), in fact unlinkability between a voter (bidder) and her vote (bid) is a desired property. Another important property for e-exams is to keep exam questions secret until the exam starts. We do not find such a property in e-voting where the candidates are previously known to the voters, and in e-auction where the goods to bid for are previously known to the bidders. Moreover, properties such as *Anonymous Marking*, meaning that an examiner can not know whose copy they are grading, evaluates to a sort of fixed-term anonymity. This property is meant to hold during the marking, but is trivially falsified when the marks are assigned to the candidates.

Verifiability has been also studied in other domains than exams, specially in voting and in auctions. In these domains formal models and definitions of security properties exist, *e.g.*, [DJL13, KTV10, KRS10b]. In voting, *individual verifiability* ensures that a voter can verify her vote has been handled correctly, that is, cast as intended, recorded as cast, and counted as recorded [BT94, HS00]. The concept of *universal verifiability* has been introduced to express that voters and non-voters can verify the correctness of the tally using only public information [CF85, BT94, Ben96]. Kremer *et al.* [KRS10b] formalize both individual and universal verifiability in the Applied π-Calculus. They also consider *eligibility verifiability*, a specific universal property assuring that any observer can verify that the set of votes from which the result is determined originates only from eligible voters, and that each eligible voter has cast at most one vote. Smyth *et al.* [SRKK10] use ProVerif to check different verifiability notions that they express as reachability properties. Verifiability for e-auction is studied by Dreier *et al.* [DJL13]. The manner in which they

express sound and complete tests for their verifiability properties has been a source of inspiration for what we present, concerning verifiability, in this thesis.

Notable notions related to verifiability are *accountability* and *auditability*. Generally speaking, a protocol is verifiable when it allows anyone (or a specific party) to verify that a certain goal is satisfied. In its turn accountability allows to identify which party is responsible when the protocol fails to meet one of its goals. Verifiability allows a party to detect that there is something wrong when the protocol fails. However, this may not be sufficient as it should be possible to specify the party that is responsible of this failure, and by this, resolve the dispute. In this sense, accountability is closely related to verifiability. Küsters *et al.* have been proposed a formal definition for accountability in [KTV10]. They also provided symbolic and computational definitions of verifiability, which they interpreted as a restricted form of accountability. However, their framework needs to be instantiated for each application by identifying relevant verifiability goals. In another hand, auditability is the quality of a protocol that stores sufficient evidence to convince an honest judge that specific properties are satisfied [GFN09]. Auditability revisits the universal verifiability defined: anyone, even an outsider without knowledge of the protocol execution, can verify the system relying only on the available pieces of evidence.

Concerning monitoring, offline monitoring of user-provided specifications over logs has been addressed in the context of several tools in the runtime verification community [BBF14]: Breach [Don10] for Signal Temporal Logic, RiTHM [NJW+13] for (variant of) Linear Temporal Logic, LogFire [Hav15] for rule-based systems, and JavaMOP [JMLR12] for various specification formalisms provided as plugins. Moreover, offline monitoring was successfully applied to other industrial case studies, *e.g.*, for monitoring financial transactions with LARVA [CP12], and monitoring IT logs with MonPoly [BCE+14].

3.3 Authentication and Privacy in Exams

We present our model of exam protocols in the Applied π-Calculus [AF01], and we propose formal definitions for the authentication and privacy properties. Then, we discuss our three case studies: the protocols by Huszti & Pethő [HP10], Giustolisi *et al.* [GLR14], and Grenoble exam.

3.3.1 Modeling Exam Protocols in The Applied π-Calculus

An exam protocol specifies the processes executed by the exam parties. The processes can exchange messages on public or private channels, create keys or fresh random values

and perform tests and cryptographic operations, which are modeled as functions on terms with respect to an equational theory describing their properties.

We distinguish the following parties: *candidates* who sit for the exam; the *examiners* who mark the answers submitted by the candidates; the *question committee*, which prepares the exam questions; the *exam authorities*, which conduct the exam, and include registrars, invigilators, exam collectors, and a notification committee. In some protocols, an authority can be responsible of two or more roles. Furthermore, we organize the exam in four phases:

- *Registration:* the exam authority (the registrar) creates a new examination and checks the eligibility of the candidate who attempts to register for it;
- *Examination:* the exam authority authenticates the candidate, and sends to her an *exam-form* that contains the exam questions. The candidate fills the form with her answer, and submits it to the exam collector;
- *Marking:* the authority distributes the form submitted by the candidate to an examiner, who in his turn evaluates and marks it;
- *Notification:* once the form has been evaluated, the mark is notified to the candidate.

Note that, each candidate can pass through the exam phases independently from the others.

Definition 3.1. (Exam Protocol). *An exam protocol is a tuple* $(C, E, Q, A_1, \ldots, A_l, \tilde{n}_p)$*, 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,* A_i*'s are the processes executed by the authorities, and* $\tilde{n}_p$ *is the set of private channel names.*

All candidates execute the same process C, and all examiners execute the same process E. However, different candidates and examiners are instantiated with different variable values, *e.g.*, keys, identities, and answers.

In our definitions, we reason about privacy using concrete instances of an exam protocol. An instance is called an *exam process*. Only honest parties are modeled in an exam process. Dishonest parties are under the control of the attacker. They are not modeled in the exam process, but subsumed by the attacker. We assume a single attacker, *i.e.*, all attackers and all dishonest parties share information and trust each other. The attacker forms the environment in which the exam process runs. He has complete control to the network, except private channels (*e.g.*, Dolev-Yao attacker [DY83a]). For instance, he can eavesdrop, remove, substitute, duplicate and delay messages that the parties are sending one another, and insert messages of his choice on the public channels. He can also manipulate data contained within his knowledge under the restriction of perfect

cryptography, for instance the only way to decrypt a ciphertext is to know the inverse key.

Definition 3.2. (Exam Process). *An* exam process *of an exam protocol given by the tuple* $(C, E, Q, A_1, \ldots, A_l, \tilde{n}_p)$ *is a closed process* $EP = \nu\tilde{n}.(C\sigma_{idc_1}\sigma_{a_1}| \ldots |C\sigma_{idc_j}\sigma_{a_j}| E\sigma_{ide_1}\sigma_{m_1}| \ldots |E\sigma_{ide_k}\sigma_{m_k}|Q\sigma_q|A_1\sigma_{dist}| \ldots |A_{l'})$*, where* $1 \leq l' \leq l$*,* $\tilde{n}$ *is the set of all restricted names, which includes some of the private channel names* $\tilde{n}_p$*;* $C\sigma_{idc_i}\sigma_{a_i}$ *are the processes executed by the candidates, the substitutions* σ_{idc_i} *and* σ_{a_i} *respectively specify the identity and the answers of the* i^{th} *candidate;* $E\sigma_{ide_i}\sigma_{m_i}$ *are the processes executed by the examiners, the substitution* σ_{ide_i} *specifies the* i^{th} *examiner's identity, and* σ_{m_i} *specifies for each possible question/answer pair the corresponding mark;* Q *is the process executed by the question committee, the substitution* σ_q *specifies the exam questions;* A_i *are the processes executed by the exam authorities that are required to be honest, the substitution* σ_{dist} *determines which answers will be submitted to which examiners for grading. Without loss of generality, we assume that* A_1 *distributes the copies to the examiners.*

Definition 3.2 equally handles examiners that are machines and those that are humans: they are both entities that mark answers. Moreover, the definition does not specify how the mark is computed. To ensure a fair marking we ought to assume the marking is deterministic: given the same answer to the same question, the examiner will attribute the same mark (at least during the same exam). This should be the case for a fair marking system, and this assumption avoids unrealistic corner cases in the definitions later on. In the real world, however, marking may not be necessarily deterministic, especially in tests with open-answer questions where same mark may given to "similar" but not identical answers. Note that, Q and A_1 can coincide when there is only one authority A: in that case, $Q\sigma_q|A_1\sigma_{dist}$ is simplified as $A\sigma_q\sigma_{dist}$.

In addition to the attacker, threats may also come from corrupted parties, who communicate with the attacker, share personal data (*e.g.*, secret keys) with him, or receive orders (*e.g.*, how to answer a question) from him. For instance, the attacker may control a legitimate user. Similarly to the approach used by Dreier in his thesis [Dre13], we model a corrupted party as P^{ch_1,ch_2} (see Definition 2.2 in Section 2.1.1). If process P is honest, then P^{ch_1,ch_2} is its corrupted version. This variant is exactly as P, but uses channels ch_1 and ch_2 to communicate with the attacker. Through ch_1, P^{ch_1,ch_2} sends all its inputs and freshly generated names (but not other channel names). From ch_2, P^{ch_1,ch_2} receives messages that can influence its behavior. Given an exam process any process P can be replaced by P^{ch_1,ch_2}. Corrupted parties, unlike dishonest parties, do not reveal (give access to) the private channels to the attacker. The messages received on private channels are revealed to the attacker but not the private channels themselves. Moreover, using P^{ch_1,ch_2} the attacker can send and receive messages on private channels the candidate has access to, if necessary. In contrast, for the attacker to play the role of

the, *e.g.*, candidate one would have to make certain private channels public (otherwise the attacker cannot be a candidate), which makes the transformation more difficult to write and may lead to false attacks depending on the protocol (but at the same time help termination). Note that, this distinction between corrupted parties and the attacker also makes corrupted parties explicit in the definition of the properties (see Section 3.3.2). Note also that in practice, notably in ProVerif this distinction is not a limitation due to the implicit replication[9].

To improve the readability of our definitions, we introduce the notation of *exam context*. An exam context $EP_I[\_]$ is the process EP without the processes of the parties included in the set I; they are replaced by "holes". We use this notation, for instance, to specify exactly the processes for candidates c_1 and c_2 without repeating the entire exam process; in that case we rewrite EP as $EP_{\{idc_1,idc_2\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}]$.

Definition 3.3. (Exam Context). *Let I be a set such that $I \subseteq I_C \cup I_E \cup \{id_Q, id_{A_1}\}$ where I_C is the set of all candidates, I_E is the set of all examiners, id_Q identifies the question committee Q, and id_{A_1} identifies the exam authority A_1. Then, given an exam process $EP = \nu\tilde{n}.(C\sigma_{idc_1}\sigma_{a_1}|\ldots|\ C\sigma_{idc_j}\sigma_{a_j}|\ E\sigma_{ide_1}\sigma_{m_1}|\ldots\ |E\sigma_{ide_k}\sigma_{m_k}|Q\sigma_q|A_1\sigma_{dist}|\ldots|A_{l'})$, we define the* exam context $EP_I[\_]$ *as follows:*

$$EP_I[\_] \equiv \ \nu\tilde{n}.(|\underset{idc_i\notin I}{C\ \sigma_{idc_i}\sigma_{a_i}}\ |\underset{ide_i\notin I}{E\ \sigma_{ide_i}\sigma_{m_i}}|\_|(Q\sigma_q)^{k_1}|(A_1\sigma_{dist})^{k_2}|A_2|\ldots|A_{l'})$$

where $k_1 = 0$ if $id_Q \in I$ and $k_1 = 1$ otherwise, and $k_2 = 0$ if $id_{A_1} \in I$ and $k_2 = 1$ otherwise[10].

In order to reason about reachability and authentication properties we consider events. Events are annotations that do not change a process behavior, but are inserted at precise locations to allow reasoning about the exam's execution. Events allow us to verify properties such as "event `bad`" is unreachable, or "on every trace event ε_2 is preceded by event ε_1". Following the technique used in [ABF07, SRKK10], events are outputs $out(\varepsilon, (M1, \ldots, M_n))$ on a special channel ε different from the ordinary channels. In accordance to ProVerif syntax, we write $\varepsilon(M1, \ldots, M_n)$ instead of $out(\varepsilon, (M1, \ldots, M_n))$, where ε is the name of the event (channel) and $M1, \ldots, M_n$ are the event's parameters.

In our model for exam protocols, we use the following events, where *idc* is the candidate identity, *ques* the question(s), *ans* the answer(s), *mark* the mark, id_{form} is an identifier of the exam-form used during marking, and *ide* is the examiner's identity:

– `register`(*idc*): is the event inserted into the registrar process at the location where candidate *idc* has successfully registered for the exam.

[9] In our case studies using ProVerif, we model the corrupted parties as dishonest ones that are subsumed by the attacker. [10] Recall, $(Q\sigma_q)^0 = 0$, and $(Q\sigma_q)^1 = Q\sigma_q$.

- `submit`$(idc, ques, ans)$: is the event inserted into the process of candidate idc in the examination phase, at the location where she sends her answer ans corresponding to the question $ques$.
- `accept`$(idc, ques, ans)$: is the event inserted into the exam collector's process in the examination phase, just after it received and accepted the exam-form ($idc, ques$, ans) from candidate idc.
- `distribute`$(idc, ques, ans, id_{form}, ide)$: is the event inserted into the authority process in the marking phase, when it assigns (distributes) the exam-form ($idc, ques, ans$) from candidate idc to the examiner ide using the identifier id_{form}.
- `attribute`$(ques, ans, mark, id_{form}, ide)$: is the event inserted into the process of the examiner ide in the marking phase, at the location where he marked the question/answer pair ($ques, ans$) identified by id_{form} with the grade $mark$.
- `notify`$(idc, mark)$: is the event inserted into the process of candidate idc in the notification phase, just after she received and accepted the grade $mark$ from the responsible authority.

Note that, id_{form} is only used to identify an exam-form during marking. This could be a pseudonym to allow anonymous marking, or simply the candidate identity if the marking is not anonymous.

3.3.2 Authentication and Privacy Properties

Usually in traditional exams, a central authority (trusted third party) authenticates candidates and checks whether they are registered for the exam, before handing the exam questions to them. Then, each candidate writes down his identity, which is often covered for anonymous marking, and his answer in an exam paper, which is collected by the authority at the end of the examination phase. Note that, each candidate also visibly writes his pseudonym. When examination phase ends, the authority hands the answers to an examiner without any modification. The examiner evaluates the answers, and assigns a mark for each pseudonym without revealing his real identity. Then, the authority maps the pseudonyms to the real identities, stores the pairs identities/marks, and publishes the pairs pseudonyms/marks.

In absence of a trusted third party, we require some guarantees concerning the correctness of the examination process, the integrity of the answers and the marks, and the privacy of the candidates and the examiners. To this end, we define property *Answer Origin Authentication* which ensures that only registered candidates can take the exam, properties *Form Authenticity* and *Form Authenticity* which ensure answers integrity and their link to the candidates until marking, and property *Form Authenticity* which

ensures marks integrity and that they are notified to the correct candidates. Moreover, we define five privacy properties: *Question Indistinguishability* ensures the secrecy of the questions before the examination phase, *Anonymous Marking* and *Anonymous Examiner* respectively ensure the anonymity of the candidates and examiners during marking, *Mark Privacy* ensures the secrecy of the marks, and *Mark Anonymity* hides the link between a candidate and his mark.

We model our authentication properties as correspondence properties among events, a well-known approach [RSG+00, RS11]. The first authentication property is *Answer Origin Authentication* which ensures that only exam-forms submitted by registered candidates are actually accepted (collected), and that one exam-form from is accepted from each candidate.

Definition 3.4. (Answer Origin Authentication). *An exam protocol ensures* Answer Origin Authentication *if for every exam process EP on every possible execution trace, each occurrence of the event* $\texttt{accept}(idc, ques, ans)$ *is preceded by a distinct occurrence of the event* $\texttt{register}(idc)$.

At examination phase, each candidate submits her exam-form with an answer, and the collector collects the forms. *Form Authorship* ensures that the content of each collected exam-form (*idc*, *ques*, and *ans*) are not modified after submission.

Definition 3.5. (Form Authorship). *An exam protocol ensures* Form Authorship *if for every exam process EP on every possible execution trace, each occurrence of event* $\texttt{accept}(idc, ques, ans)$ *is preceded by a distinct occurrence of event* $\texttt{submit}(idc, ques, ans)$.

Similarly, *Form Authenticity* ensures that the content of each exam-form is not modified after the collection and until after the form is marked by an examiner.

Definition 3.6. (Form Authenticity). *An exam protocol ensures* Form Authenticity *if for every exam process EP on every possible execution trace, each occurrence of the event* $\texttt{attribute}(ques, ans, mark, id_{form}, ide)$ *is preceded by a distinct occurrence of the events* $\texttt{accept}(idc, ques, ans)$ *and* $\texttt{distribute}(idc, ques, ans, id_{form}, ide)$.

At notification phase, the candidate should receive the mark which was assigned by the examiner to her answer. We call this property *Mark Authenticity*.

Definition 3.7. (Mark Authenticity). *An exam protocol ensures* Mark Authenticity *if for every exam process EP on every possible execution trace, each occurrence of the event* $\texttt{notify}(idc, mark)$ *is preceded by a distinct occurrence of the events* $\texttt{attribute}(ques, ans, mark, id_{form}, ide)$ *and* $\texttt{distribute}(idc, ques, ans, id_{form}, ide)$.

Note that *Mark Authenticity* ensures that the candidate is notified with the mark delivered by the examiner on the answer distributed for him by the authority. This answer may

be different from that submitted by the candidate. Only if also *Form Authorship* and *Form Authenticity* hold then the candidate can be sure that the assigned and submitted answers are identical. Moreover, *Mark Authenticity* does not guarantee that the mark is computed correctly. However in Section 3.4.1, we define Marking Correctness individual verifiability which allows a candidate to verify that his mark was computed correctly. We also define Marking Correctness universal verifiability, in Section 3.4.2, which allows any auditor to check the correctness of the delivered marks.

We model our privacy properties as observational equivalence, a standard choice for such kind of properties [RS01, RS11]. We use the *labeled bisimilarity* ($\approx_l$) to express the equivalence between two processes [AF01]. Informally, two processes are equivalent if an observer has no way to tell them apart. In this section, we use the notation $EP|_{\texttt{ph}}$ which denotes the process EP without the code that follows the phase `ph`. The first privacy property *Question Indistinguishability* says that questions are kept secret until the exam starts.

Definition 3.8. (Question Indistinguishability). *An e-exam protocol ensures* Question Indistinguishability *if for any e-exam process EP that ends with the registration phase, any questions q_1 and q_2, we have that:*

$$EP_{\{id_Q\}}[Q\sigma_{q_1}]|_{\texttt{reg}} \approx_l EP_{\{id_Q\}}[Q\sigma_{q_2}]|_{\texttt{reg}}$$

Question Indistinguishability states that two processes with different questions have to be observationally equivalent until the end of the registration phase. This prevents the attacker from obtaining information about the exam questions before the examination phase starts. This property requires the question committee to be honest; otherwise the property is trivially violated since the committee reveals the questions to the attacker. However, it is particularly interesting to consider corrupted candidates, as they might be interested in obtaining the questions in advance. We can do this by replacing honest candidates with corrupted ones. For example, if we assume that candidate idc_1 is corrupted, we obtain:

$$EP_{\{idc_1,id_Q\}}[(C\sigma_{idc_1}\sigma_{a_1})^{ch_1,ch_2}|Q\sigma_{q_1}]|_{\texttt{reg}} \approx_l EP_{\{idc_1,id_Q\}}[(C\sigma_{idc_1}\sigma_{a_1})^{ch_1,ch_2}|Q\sigma_{q_2}]|_{\texttt{reg}}$$

The same technique can be employed to add more corrupted candidates. Note that, this can also be applied to all the following privacy properties to add corrupted parties if required. The next property ensures that the marking process is done anonymously, *i.e.*, that two instances where candidates swap their answers cannot be distinguished until after the end of the marking phase. This may be desirable to ensure fairness of the grading, and is a requirement in some exam settings (at some universities or for competitive examinations).

Definition 3.9. (Anonymous Marking). *An e-exam protocol ensures* Anonymous Marking *if for any e-exam process EP that ends with the marking phase, any two candidates idc_1 and idc_2, and any two answers a_1 and a_2, we have that:*

$$EP_{\{idc_1,idc_2\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}]|_{\texttt{mark}} \approx_l EP_{\{idc_1,idc_2\}}[C\sigma_{idc_1}\sigma_{a_2}|C\sigma_{idc_2}\sigma_{a_1}]|_{\texttt{mark}}$$

Anonymous Marking ensures that the process where idc_1 answers a_1 and idc_2 answers a_2 is equivalent to the process where idc_1 answers a_2 and idc_2 answers a_1. This prevents the attacker from obtaining the identity of the candidate who submitted a certain answer before the marking phase ends. For this property, it is interesting to consider corrupted examiners. It can be done using the same technique employed for corrupted candidates above. We can also have some corrupted candidates, however the candidates idc_1 and idc_2 who are assigned the two different answers have to be honest – otherwise the property can be trivially violated by one of them revealing her answer to the attacker. Note that by controlling a candidate, the attacker could be able to compromise privacy by trying to relate the corrupted candidate's answer to the targeted candidate's answer. To prevent bribing or coercion of the examiners, it might be interesting to ensure their anonymity, so that no candidate knows which examiner marked her copy. This is ensured by *Anonymous Examiner*.

Definition 3.10. (Anonymous Examiner). *An e-exam protocol ensures* Anonymous Examiner *if for any e-exam process EP, any two candidates idc_1, idc_2, any two examiners ide_1, ide_2, and any two marks m_1, m_2, we have that:*

$$EP_{\{idc_1,idc_2,ide_1,ide_2,id_{A_1}\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}|E\sigma_{ide_1}\sigma_{m_1}|E\sigma_{ide_2}\sigma_{m_2}|A_1\sigma_{dist_1}]$$
$$\approx_l$$
$$EP_{\{idc_1,idc_2,ide_1,ide_2,id_{A_1}\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}|E\sigma_{ide_1}\sigma_{m_2}|E\sigma_{ide_2}\sigma_{m_1}|A_1\sigma_{dist_2}]$$

where σ_{dist_1} attributes the exam-form of candidate idc_1 to examiner ide_1 and the exam-form of candidate idc_2 to examiner ide_2, and σ_{dist_2} attributes the exam-form of candidate idc_1 to examiner ide_2 and the exam-form of candidate idc_2 to examiner ide_1.

Anonymous Examiner ensures that a process in which examiner ide_1 grades the exam-form of candidate idc_1 and examiner ide_2 grades that of candidate idc_2 cannot be distinguished from a process in which ide_1 grades the exam-form of idc_2 and ide_2 grades that of idc_1. Note that to ensure that in both cases the candidates receive the same mark, we also have to swap σ_{m_1} and σ_{m_2} between the examiners. Similar to *Anonymous Marking*, this property prevents the attacker from obtaining the identity of the examiner who marked a certain answer. Note that, *Anonymous Examiner* requires that the examiners ide_1 and ide_2 are honest, otherwise it will trivially violated by one of them revealing the

mark he gave. We can again include corrupted candidates as they might be interested in finding out which examiner marked their copies. Note also that, if all the exam-forms are assigned to the same examiner to mark them (or simply if we have only one examiner), then all the candidates may know whom is going to evaluate their exam-forms. To avoid such cases, it is required to have properties that ensures the secrecy of the examiners' number, and the number of exam-forms assigned for each examiner. However, in practice a countermeasure for such issues is to have at least two examiners and to uniformly distribute the exam-forms between them.

In some exams settings the marks have to remain private. This is ensured by *Mark Privacy*. *Mark Privacy* guarantees that two processes where the examiner *ide* assigns for the same answer (entailed by the same context EP) two different marks (specified by the substitutions σ_{m_1} and σ_{m_2}) cannot be distinguished from each other.

Definition 3.11. (Mark Privacy). *An e-exam protocol ensures* Mark Privacy *if for any e-exam process EP, any examiner ide, any two substitutions σ_{m_1} and σ_{m_2}, we have that:*

$$EP_{\{ide\}}[E\sigma_{ide}\sigma_{m_1}] \approx_l EP_{\{ide\}}[E\sigma_{ide}\sigma_{m_2}]$$

Note that, as σ_{m_1} and σ_{m_2} are different then at least one of the answers assigned to the examiner *ide* is attributed with different marks. Depending on the exam policy this can be an optional property since some exams system may publicly disclose the marks of the candidates. However, the intuition here is that candidate's performance should not be known to any other candidate. Note that our formalization does not contradict our hypothesis that the marking within one exam is deterministic, as we consider two different instances of an exam (notably differing in the marking). Again, we can assume that some candidates are corrupted and try to find out the marks of their colleagues, or that an examiner tries to find out the mark achieved by a candidate. The candidate who is assigned the two different marks has to be honest – otherwise the property is violated by her revealing her mark to the attacker. Similarly the examiner assigning the marks has to be honest, otherwise he can reveal the mark himself. Note that, considering corrupted candidates allows us to capture attacks such as copying someone else's answers. For instance, the attacker may copy a honest candidate's answer and order a corrupted candidate to submit the same answer. Then, the attacker can compromise the privacy of the honest candidate's mark in case of deterministic marking[11] (as both the corrupted and honest candidates will get the same mark).

The previous definition of *Mark Privacy* ensures that the attacker cannot know the mark of a candidate. A weaker variant of *Mark Privacy* is *Mark Anonymity*, *i.e.*, the attacker might know the list of all marks, but is unable to associate a mark to its

[11] Similar to the attack found against 2.0 voting system by Cortier and Smyth [CS13].

corresponding candidate. This is often the case in practice, where a list of pseudonyms (*e.g.*, student numbers) and marks is published.

Definition 3.12. (Mark Anonymity). *An e-exam protocol ensures* Mark Anonymity *if for any e-exam process EP, any candidates idc_1, idc_2, any examiner ide, any answers a_1, a_2 and a distribution σ_{dist} that assigns the answers of both candidates to the examiner ide, and two substitutions σ_{m_b} and σ_{m_c} which are identical, except that σ_{m_b} attributes the mark m_{a_1} to the answer a_1 and m_{a_2} to a_2, whereas σ_{m_c} attributes m_{a_2} to the answer a_1 and m_{a_1} to a_2, we have that:*

$$EP_{\{idc_1,idc_2,ide,id_{A_1}\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}|E\sigma_{ide}\sigma_{m_b}|A_1\sigma_{dist}]$$
$$\approx_l$$
$$EP_{\{idc_1,idc_2,ide,id_{A_1}\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}|E\sigma_{ide}\sigma_{m_c}|A_1\sigma_{dist}]$$

Mark Anonymity states that if an examiner *ide*, who is assigned the same answers a_1 and a_2 to mark as σ_{dist} is unchanged, swaps the marks between these answers, then the two situations cannot be distinguished by the attacker. This means that a list of marks can be public, but the attacker must be unable to link the marks to the candidates. Again, we can consider corrupted parties, but this definition requires the two concerned candidates and the concerned examiner to be honest. Otherwise they can simply reveal the answer and the associated mark, which allows to distinguish both cases. A protocol that ensures *Mark Privacy* also ensures *Mark Anonymity*. In fact, σ_{m_b} and σ_{m_c} are special cases of σ_{m_1} and σ_{m_2}.

Theorem 3.1. *If an exam protocol satisfies* Mark Privacy*, it also satisfies* Mark Anonymity*.*

Proof. Suppose that *Mark Privacy* is satisfied, then for any exam context $EP_{\{ide\}}$, any two mark substitutions σ_{m_1} and σ_{m_2}, we have that: $EP_{\{ide\}}[E\sigma_{ide}\sigma_{m_1}] \approx_l EP_{\{ide\}}[E\sigma_{ide}\sigma_{m_2}]$.

Let $EP'[\_] \equiv EP_{\{idc_1,idc_2,ide,id_{A_1}\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}|A_1\sigma_{dist}|\_]$ be an exam context, where $EP_{\{idc_1,idc_2,ide,id_{A_1}\}}[\_]$ is any exam context, and σ_{dist} distribute the answers a_1 and a_2 to an examiner *ide*. Also let σ_{m_b} be a substitution that assigns a mark m_{a_1} to a_1 and a mark m_{a_2} to a_2, whereas σ_{m_c} be a substitution that assigns a mark m_{a_2} to a_1 and a mark m_{a_1} to a_2. Then, as *Mark Privacy* is satisfied, we have that: $EP'[E\sigma_{ide}\sigma_{m_b}] \approx_l EP'[E\sigma_{ide}\sigma_{m_c}]$. Hence

$$EP_{\{idc_1,idc_2,ide,id_{A_1}\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}|E\sigma_{ide}\sigma_{m_b}|A_1\sigma_{dist}]$$
$$\approx_l$$
$$EP_{\{idc_1,idc_2,ide,id_{A_1}\}}[C\sigma_{idc_1}\sigma_{a_1}|C\sigma_{idc_2}\sigma_{a_2}|E\sigma_{ide}\sigma_{m_c}|A_1\sigma_{dist}]$$

Therefore, *Mark Anonymity* is satisfied. □

3.3.3 Case Studies

We analyze using ProVerif[12] the authentication and privacy properties of three exam protocols: the protocol by Huszti & Pethő [HP10], the protocol by Giustolisi *et al.* [GLR14], and the protocol of Grenoble exam.

3.3.3.1 Protocol by Huszti & Pethő

We first analyze the protocol by Huszti & Pethő [HP10] (in short, H&P protocol). This protocol aims to ensure authentication and privacy for e-exams in presence of corrupted candidates, examiners, and exam authorities; the guarantees are argued only informally in [HP10]. Notably from a point of view of the protocol paper [HP10], all arguments supporting privacy rely on the reliability of a single component, the *reusable anonymous return channel*, or RARC [GJ03]. The description of RARC channel is outlined below.

Reusable Anonymous Return Channel (RARC). A RARC implements anonymous two-way conversations. A sender posts a message to a recipient and the RARC ensures its anonymity; in its turn, the recipient can reply to that message without knowing nor learning the sender's identity, sure that the RARC will dispatch it to actual sender. RARC ensures the anonymity of the messages, and the entire conversation remains untraceable to an external attacker, but it does not guarantee the secrecy of the messages [GJ03]. A RARC is implemented by a re-encryption mix networks, first proposed by Chaum [Cha81]. The mix servers jointly generate and share an ElGamal [ElG85] key pair (PK_{MIX}, SK_{MIX}) and a pair of public/private signing keys (SPK_{MIX}, SSK_{MIX}). The sender A and the receiver B also hold ElGamal public/private key pairs, (PK_A, SK_A) and (PK_B, SK_B) respectively. A and B are represented by ID_A and ID_B, identity tags which can be for example A's and B's email addresses. To send the message m to B, the sender A submits to the mix networks the tuple $Mix(m, A, B)$ which denotes $(\{ID_A, PK_A\}_{PK_{MIX}}, \{m\}_{PK_{MIX}}, \{ID_B, PK_B\}_{PK_{MIX}})$[13] and proves knowledge of $\{ID_A, PK_A\}$ and of $\{ID_B, PK_B\}$. The proofs of knowledge are claimed to impede the attacker from decrypting the triplets by using the mix networks as an oracle (we falsify this claim below). The mix networks waits to collect more triplets and shuffles them. Then, it adds a checksum to the triplets, which is supposed to vouch for integrity (again, we disprove this claim below). The message m is then re-encrypted with the public key of B using a switching encryption keys technique. The mix networks signs the encrypted public key of A. Thus B receives $(sign(\{ID_A, PK_A\}_{PK_{MIX}}, SK_{MIX}), \{m\}_{PK_B})$ where $sign(x, sk)$ is message x plus the signature with the secret key sk. Then B replies to A with a new message m' by sending to the mix networks$(Mix(m', B, A), sign(\{ID_A, PK_A\}_{PK_{MIX}}, SK_{MIX}))$

[12] All the verification code used in this thesis is available on line at the link http://www-verimag.imag.fr/ plafourc/kassem-thesis-code.zip

[13] $\{x\}_{PK}$ denotes the public-key encryption of x by the key PK.

and proving only knowledge of $\{ID_B, PK_B\}$. The mix networks checks the proof and the signature, and then processes the tuples like a normal message.

Protocol Description. A symbolic representation of the H&P protocol is depicted in Figure 3.1. Note that all messages, except step 2, are sent via RARC. H&P protocol relies upon different cryptographic building blocks. The ElGamal cryptosystem [ElG85] is used to provide parties with public/private key pairs. A RARC implements anonymous two-way communication. A network of servers (NET) provides a timed-release service. The NET creates and revokes a candidate's pseudonym. More precisely, the NET's contribution to the pseudonym is shared among the servers using the threshold Shamir secret sharing system [Sha79]. At notification, a subset of the NET servers use their shares to recover the secret and de-anonymize the candidate: the exam authority can so associate the answer with the corresponding candidate. To avoid plagiarism, the protocol assumes that no candidate reveals his private key to another candidate, and that invigilators supervise candidates during the examination.

The original protocol has the following phases: setup, registration, exam (combines examination and marking), and grading (notification). To match this structure with our exam model, we merge the setup and registration but we split between candidate registration and examiner registration for better readability. We also split between examination and marking phases as they are considered one phase in the original paper, called exam.

Examiner Registration: the exam authority (EA) publishes the public parameters (h, g)[14], which identify a new examination (step 1). The question committee (QC) then signs and sends to EA the questions and the starting time of the examination phase $time_1$ encrypted with the public key of the RARC mix networks PK_{MIX} (step 2). The mix networks forwards the message only when the examination begins. Thus, even the exam authority cannot learn the questions. Then, a pseudonym of the examiner is jointly created by the exam authority and the examiner (steps 3-4). Note that, g_E is a part of examiner's public key, we have $PK_E = g_E^{SK_E}$. The examiner (*i.e.*, the verifier) verifies the correctness of the pseudonym using an interactive zero-knowledge proof on the equality of the discrete logarithms ZKP_{eq} with the exam authority as the prover (steps 5-6). This verification requires several exchanges of messages between the two parties, which is denoted by ($\leftrightarrow$). Then, the examiner sends his pseudonym (t, q, q') to the exam authority (step 7), and proves the knowledge of his secret key using an interactive zero-knowledge proof ZKP_{sec} (step 9). Finally, exam authority stores examiner pseudonym, transcript of ZKP_{sec}, the identity of the examiner encrypted with the mix networks public key, and the subject of the exam (step 10).

[14] Note that, $s \in \mathbb{Z}_Q$ and $s \in \mathbb{G}_Q$, where $\mathbb{G}_Q$ denotes $\mathbb{Z}_P^*$'s multiplicative subgroup of order Q such that P and Q are large primes and $Q|(P-1)$.

Candidate Registration: the registration of the candidate (C) slightly differs from the registration of an examiner. The candidate pseudonym is jointly calculated by the exam authority, the candidate, and also the NET to provide anonymity for the candidates. The NET stores the secret values used for the pseudonym generation, which can be used to de-anonymize the candidate after the examination has finished. The involvement of the NET is needed to revoke the anonymity of the candidate at notification. The candidate pseudonym is initially calculated by the exam authority (step 11), then anonymized by the NET (steps 13), and finally computed by the candidate using his private key (step 15). Again, the candidate finally verifies the correctness of his pseudonym using ZKP_{eq} (steps 16).

Examination: the candidate sends his pseudonym *via* the RARC to the exam authority and proves the knowledge of his private key (steps 17-19). Then, the exam authority checks whether the candidate is registered for the examination, and sends to him the questions signed by the question committee (step 20). The candidate sends his answer, again *via* the RARC (steps 21), thus the exam authority cannot learn the answer. The exam authority replies with a receipt which consists of the hash (H) of all parameters seen by the exam authority during the examination, the transcription $trans_C$ of ZKP_{sec}, and the time when the answer was submitted $time_2$ (step 22).

Marking: the exam authority chooses an examiner who is eligible for the examination, and forwards him the answer *via* the RARC (step 23). Note that, ID_{EAP} is a specially generated identification exam tag, and that EA stores $(\{ID_E, PK_E\}_{PK_{MIX}})$. Then, the examiner assigns a mark to the answer (step 24), and authenticates them with the transcript $verzkp$ of a non-interactive zero-knowledge proof of equality of discrete logarithm of $(H(mark, answer), [H(mark, answer)]^{SK_E}, q, q')$ (step 25).

Notification: when all the answers are marked, the NET de-anonymizes the pseudonyms linked to the answers (step 26-28). The exam authority publishes the marks (step 29).

Formal Analysis. The equational theory depicted in Figure 3.2 models the cryptographic primitives used within the H&P protocol. It includes the well-known models for probabilistic encryption (functions `pk`, `enc`, `dec`) and digital signature (functions `sign`, `getmess`, and `checksign`). We use the equation $\texttt{exp}(\texttt{exp}(\texttt{exp}(g, x), y), z) = \texttt{exp}(\texttt{exp}(\texttt{exp}(g, y), z), x)$ to model a commutative feature of the exponentiation function `exp`. This feature is needed to captures the checks by EA in steps 8 and 18 of Figure 3.1. Inspired by Backes *et al.* [BMU08], we model the ZKP of knowledge of a secret exponent as two functions: `zkp_proof` for proof, and `zkp_sec` for verification. The function $\texttt{zkp\_proof}(public; secret)$ takes as arguments a secret e' and public parameters $exp(g, e)$ and $exp(exp(g, e), e')$. It can be constructed only by the prover who knows the secret parameter. The verification function $\texttt{zkp\_sec}(\texttt{zkp\_proof}(public; secret), verinfo)$ takes

Examiner Registration
1- EA publishes g and $h = g^s$
2- $QC \rightarrow EA : \{sign((question, time_1), SSK_{QC})\}_{PK_{MIX}}$
3- EA checks E eligibility, and calculates $\tilde{q} = PK_E^s$
4- $EA \rightarrow E : (\tilde{q}, g_E)$
5- E calculates $q' = \tilde{q}^{\alpha}$, $t = g_E^{\alpha}$, and $q = t^{SK_E}$, where α is randomly chosen.
E pseudonym is (t, q, q')
6- $EA \leftrightarrow E : ZKP_{eq}((q, q'), (g, h))$
7- $E \rightarrow EA : (t, q, q', subject)$, where $subject$ is exam name
8- EA checks $q^s = q'$
9- $E \leftrightarrow EA : ZKP_{sec}(SK_E)$
10- EA stores (t, q, q'), ZKP_{sec} data plus $\{ID_E, PK_E\}_{PK_{MIX}}$ and $subject$

Candidate Registration
11- EA checks C eligibility, and calculates $\tilde{p} = PK_C^s$
12- $EA \rightarrow NET : (\tilde{p}, g_C)$
13- NET calculates $p' = \tilde{p}^{\Gamma}$, and $r = g_C^{\Gamma}$, where Γ is randomly chosen.
The NET then stores the $time$ when g_C can be revealed, $\tilde{p}$, g_C,
and y_i which is its share applied for secret sharing [Sha79]
14- $NET \rightarrow C : (r, p')$
15- C calculates $p = r^{SK_C}$
16- $EA \leftrightarrow C : ZKP_{eq}((p, p'), (g, h))$
Then C pseudonym is (r, p, p')

Examination
17- $C \rightarrow EA : (r, p, p', subject)$
18- EA checks $p^s = p'$
19- $C \leftrightarrow EA : ZKP_{sec}(SK_C)$
20- $EA \rightarrow C : (sign(question, SSK_{QC}), time_1)$
21- $C \rightarrow EA : (r, p, \{answer\}_{PK_{MIX}}, time_2)$
22- $EA \rightarrow C : H(r, p, p', subject, trans_C, question, time_1, time_2, \{answer\}_{PK_{MIX}})$

Marking
23- $EA \rightarrow E : (\{ID_{EAP}, PK_{EAP}\}_{PK_{MIX}}, answer, \{ID_E, PK_E\}_{PK_{MIX}})$
24- $E \rightarrow EA : (mark, H(mark, answer), [H(mark, answer)]^{SK_E}, verzkp)$
25- $E \leftrightarrow EA : ZKP_{eq}(H(mark, answer), [H(mark, answer)]^{SK_E}), (t, q))$

Notification
26- $EA \rightarrow NET : p'$
27- NET calculates $p' = \tilde{p}^{\Gamma}$
28- $NET \rightarrow EA : \{(p', \tilde{p})\}_{PK_{EA}}$
29- EA publishes : $(mark, H(mark, answer), [H(mark, answer)]^{SK_E}, verzkp)$

FIGURE 3.1: A symbolic representation of the H&P protocol. All messages, except step 2, are sent *via* RARC.

as arguments the proof function and the verification parameters $verinfo$ (*i.e.*, $exp(g, e)$ and $exp(exp(g, e), e')$). The verifier only accepts the proof if the relation between $verinfo$ and $secret$ is satisfied. Similar to `zkp_proof`, the function `xproof`$(public; secret)$ takes a secret e and public parameters $p, p', g,$ `exp`(g, e). In its turn `check_proof` takes as

arguments the `xproof` function and $verinfo$. Note that, in addition to the function `check_proof`, we use tables to support the model for the ZKP of the equality of discrete logarithms. This is due to the difficulties of ProVerif when dealing with associativity of multiple exponents, which is used in the H&P protocol (see Figure 3.1). Particularly, we use tables to allow the candidate to verify that $\tilde{p}$ and p' have been correctly generated respectively by the exam authority and the NET (see steps 11 and 13 in Figure 3.1). This approach is mainly needed to let ProVerif terminate for *Mark Privacy* and *Mark Anonymity*. It is sound because it limits the attacker capability to generate fake ZKPs, as he cannot write and read ProVerif's table. Nevertheless, ProVerif still finds counterexamples that falsify these two properties, as shall we see later.

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

$$\begin{aligned}
\mathtt{dec}(\mathtt{enc}(m, pk(k), r), k) &= m \\
\mathtt{getmess}(\mathtt{sign}(m, k)) &= m \\
\mathtt{checksign}(\mathtt{sign}(m, k), pk(k)) &= m \\
\mathtt{exp}(\mathtt{exp}(\mathtt{exp}(g, x), y), z) &= \mathtt{exp}(\mathtt{exp}(\mathtt{exp}(g, y), z), x) \\
\mathtt{zkp_sec}(\mathtt{zkp_proof}(exp(g, e), exp(exp(g, e), e'), e'), exp(g, e), exp(exp(g, e), e')) &= true \\
\mathtt{check_proof}(\mathtt{xproof}(p, p', g, \mathtt{exp}(g, e), e), p, p', g, \mathtt{exp}(g, e)) &= true
\end{aligned}$$

FIGURE 3.2: Equational theory for our model of H&P protocol.

Attack on RARC. First we analyze the RARC alone and show that there are attacks on anonymity and privacy. ProVerif shows that the RARC fails to guarantee both secrecy of messages and anonymity of sender and receiver identities, which is its main purpose inside the H&P protocol. We refer the triplet (c_1, c_2, c_3) as the encrypted messages that A submits to the mix networks when she wants to send a message to B. From the description of RARC given at the beginning of this section, we recall that c_1 encrypts the A's public key, c_2 encrypts the message to B, and c_3 encrypts the B's public key. All ciphertexts are encrypted with the mix networks's public key. The attacker uses the RARC as a decryption oracle, letting the RARC reveal any of the plaintexts. The attack works as follows. The attacker chooses one of the three ciphertexts (depending on whether he wants to target the contents of the message, or the identities of the sender and receiver) and submits this as a new message. For example, if the attacker targets $c_1 = \{ID_A, PK_A\}_{PK_{MIX}}$, he resubmits c_1 as a new encrypted message, which means that

$c'_2 = c_1$ in the new triplet (see Figure 3.3). He can leave the encryption of the senders key and the proof concerning the key unchanged, but replaces the encryption of the receiver's key with a public key PK_I for which he knows the corresponding secret key SK_I. In our example this means $c'_3 = \{ID_I, PK_I\}_{PK_{MIX}}$. The attacker can also provide the necessary proof of knowledge of plaintext, since he knows this plaintext. The RARC then mixes the input messages, and sends the encryption of the message under the receiver's public key to the receiver. In our example the attacker receives $d = \{ID_A, PK_A\}_{PK_I}$. Since the attacker knows the secret key SK_I he can obtain the original message. In our example he gets ID_A, the identity of the sender which should have remained anonymous. Since the attacker can substitute any of the items in the triplet as the new message, the RARC does neither ensure secrecy of the messages nor the anonymity of the sender or the receiver. Note that, the checksum meant to guarantee the integrity of the triplet is only added after the submission of the message and is only used inside the mix networks. Hence, the checksum does not prevent the attacker from submitting a modified triplet. Even if it were added before, it would not prevent the attack as the knowledge of the ciphertexts is sufficient to compute the checksum. Note that, the RARC was originally designed to withstand a passive attacker that however can statically corrupts parties [GJ03]. This is not realistic in the e-exam setting where corrupted parties could actively try to cheat.

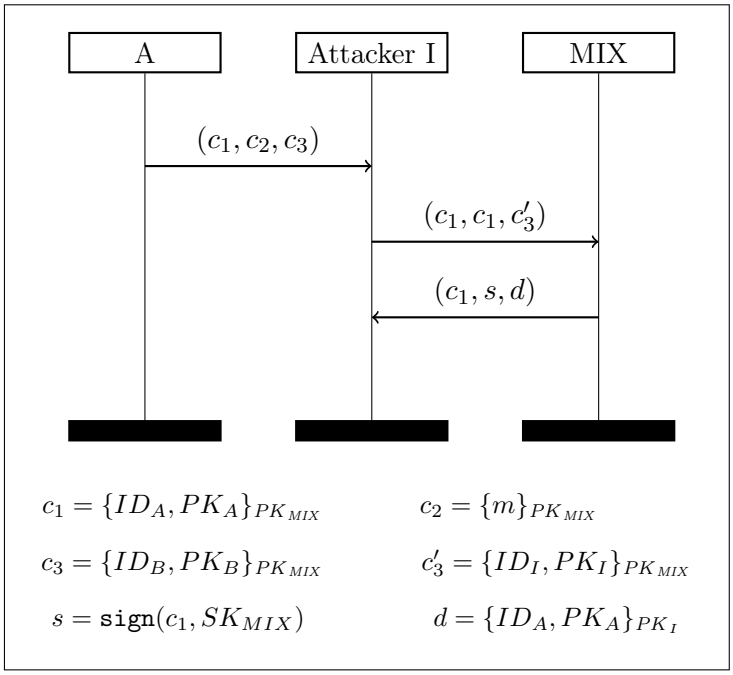


FIGURE 3.3: A symbolic representation for an attack on RARC.

All properties fail with such a RARC. However, even if we replace this RARC with an ideal implementation – which, according to the RARC original requirements [GJ03], ensures anonymity of senders and receivers but not message secrecy, implemented as an anonymous channel in ProVerif – the H&P protocol does not satisfy any of our properties.

The next paragraphs details the findings, and Table 3.1 summarizes the results found using ProVerif.

Authentication properties. We verified the authentication properties without and with an ideal RARC. All the following counterexamples remain valid in both cases. ProVerif finds a counterexample for *Answer Origin Authentication* where the attacker can create a fake pseudonym that allows him to take part in an exam for which he 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 timed-release service. The attacker generates his own secret key SK_A, and calculates an associate pseudonym, which sends to the exam authority. The exam authority successfully verifies the received data and that the attacker knows SK_A, thus the exam authority accepts the answer. Regarding *Form Authorship*, ProVerif shows the same attack trace that falsifies Answer Origin Authentication. In fact, the exam authority may collect an exam-form where the pseudonym is exchanged with one chosen by the attacker.

ProVerif also shows that the H&P protocol does not ensure *Form Authenticity*, because there is no mechanism that allows the examiner to check whether the answers have been forwarded by the exam authority. Even if the original RARC is used and the answer is encrypted with the public key of the mix networks, this does not guarantee that the exam authority actually sent the message. Regarding *Mark Authenticity*, ProVerif provides a counterexample in which the attacker can forward any answer to any examiner, even if the answer was not collected by the exam authority. Moreover, the attacker can notify the candidate by himself with a mark of his choice.

Privacy properties. ProVerif finds an attack trace on *Question Indistinguishability*. This is because the attack on the RARC exposes the message and the identities of the sender and receiver. As the questions are sent through the RARC, the attacker can obtain them. Since the candidate's answer is also sent through the RARC, *Anonymous Marking* does not hold: the answer can be linked to its corresponding sender. The protocol ensures neither *Mark Privacy* nor *Anonymous Examiner*, as the marks are also sent through the RARC. Hence, they can be decrypted and the examiner can be identified.

We checked the H&P protocol in ProVerif assuming ideal RARC. In this case, ProVerif shows an attack for each property. *Anonymous Examiner* can be violated because the attacker can track which examiner accepts the ZKP when receiving the partial pseudonym, and then associate to the examiner the answer that the latter grades. Moreover, a similar attack on *Anonymous Marking* remains: the attacker can check whether a candidate accepts the ZKP to associate him with a pseudonym, and then identify his answer. *Mark Privacy* fails because the examiner sends the mark to the exam authority *via* the RARC, which does not ensure secrecy. Finally, ProVerif shows that the H&P protocol does

not satisfy *Mark Anonymity*: the attacker can track which pseudonym is assigned to a candidate and the mark is not secret, and link a candidate to the assigned mark.

Fixing Authentication. We believe that, in H&P protocol, authentication is compromised due to inaccuracies in the protocol design, whereas most of attacks that invalid privacy are due to compromising secrecy and anonymity over the RARC. Note that, even when assuming an ideal RARC ensuring anonymity, we still have attacks on all properties. Thus, we think that fixing the RARC is not sufficient to ensure privacy – the protocol requires fundamental changes. However, we propose four simple modifications to the H&P protocol in order to achieve a set of authentication properties. In particular, we prove in ProVerif that the modified protocol achieves *Answer Origin Authentication*, *Form Authenticity*, and *Mark Authenticity*. However, the protocol fails to satisfy *Form Authorship* even after applying our fix. We found no easy solution for *Form Authorship* as the protocol sees no signatures for candidates, and RARC does not guarantee authentication.

Concerning *Candidate registration*, we observe that EA and NET do not need to communicate anonymously *via* RARC, as the original protocol prescribes. Conversely, they both need to authenticate each other messages to avoid considering attacker message injections. Thus, the first modification consists on the NET receiving the partial pseudonyms generated by EA *via* a secure channel instead *via* a RARC. In doing so, the attacker cannot use the NET to generate fake pseudonyms. As second modification we let NET send *via* secure channel the eligible pseudonyms to the EA, who, in doing so, can generate ZK proofs of equality of discrete logarithm to eligible pseudonyms only. The EA can also store the eligible pseudonyms, which can be checked at examination before accepting a test from a candidate. Concerning *Marking*, we note that the examiner cannot verify whether a test has been sent by the EA. Since the anonymity requirement is on the examiner but not on the EA, the latter can sign the test. Thus, the third modification consists on EA signing the test prior to forward it to the chosen examiner, who authenticates the signature. The last issue concerns the form identifier the EA affixes to the test before forwarding it to the examiner. Since the candidate is unaware of such identifier, the attacker can notify him any other examiner's mark. The forth modification sees the EA adding the form identifier to the candidate's submission receipt, which is also signed by the EA. Also the examiner adds the form identifier to the marking receipt, so the candidate can verify whether he has been notified with the correct mark.

3.3.3.2 Protocol by Giustolisi *et al.*

In this section, we analyze the protocol by Giustolisi *et al.* [GLR14], also called Remark! protocol. We first describe the protocol before presenting the results of our analysis.

Property	Result	Time
Answer Origin Authentication	×	26 s
*Answer Origin Authentication**	✓ (E,EA,C,NET)	3 s
Form Authorship	×	3 s
Form Authenticity	×	33 s
*Form Authenticity**	✓ (E,EA,C,NET)	3 s
Mark Authenticity	×	52 s
*Mark Authenticity**	✓ (E,EA,C,NET)	4 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 3.1: Results of our analysis on the formal model of the H&P protocol. The parties which are assumed to be honest for the result to hold are in brackets. (×) indicates that the property does not hold despite all parties being honest. (*) is the result after applying our fixes.

Protocol Description. The Remark! protocol mainly relies on several servers (NET) that implement an *exponentiation mix networks* [HS11]. The specialty of exponentiation mix networks is that each server blinds its entries by a common exponent value r_i. That is, given an input X to the mix networks, it outputs X^r where $r = \prod_i r_i$ is the product of the secret exponent values of the servers. Note that only one of the mix networks servers is required to be honest. Remark! protocol particularly uses the exponentiation mix networks to create the pseudonyms for the candidates and examiners at registration (see below). The mix networks is also used at notification to revoke the candidates pseudonyms and retrieve their identities. Remark! protocol also uses a bulletin board[15] to publish some data such as the pseudonyms, the test questions and the receipts of test submissions.

Remark! protocol has the following parties: exam authority (EA, called manager in the original paper), mix networks servers (NET), examiners (E), and candidates (C). Each party is assumed to have a pair of public/private key with a common generator g, *i.e.*, the private key x and the public key $y = g^x$. In the following, we present the Remark! protocol within the four exam phases. A symbolic representation of Remark! protocol is depicted in Figure 3.4.

Registration: at registration, the NET creates the pseudonyms for the candidates and examiners without involving any of them. The list of eligible candidates' public keys is sent as a batch to the NET. For each candidate (C), the NET calculates the pseudonyms $\overline{PK}_C$ by raising the initial public key $PK_C = g^{SK_C}$ to a common secret value r_c, where SK_C is the candidate's secret key (Step 1). Then, along with the pseudonym

[15] A public append-only memory.

$\overline{PK}_C = PK_C^{r_c} = (g^{SK_C})^{r_c}$, the NET publishes on the bulletin board (BB) the new generator $h_c = g^{r_c}$ (all signed by its secret key SK_{NET}, Step 2). The candidates can identify their own pseudonyms by raising h to their secret key x, *i.e.*, $\overline{PK}_C = h_c^{SK_C}$ (Step 3). The pseudonym $\overline{PK}_C$ eventually serve as public keys which corresponds to the secret key SK_C with the new generator h_c (as $\overline{PK}_C = h_c^{SK_C}$). This is to allow parties to communicate anonymously. The NET creates the pseudonym of each examiner (E) in the same way (Steps 4-6). Note that, two different batches are used for candidates and examiners because only the identities of candidates are revealed at notification.

Examination: the exam authority signs and encrypts the test questions with the candidate's pseudonym and publishes them on the bulletin board (Step 7). Each candidate submits his answer, which is signed with the candidate's private key (but using the generator h instead of g) and encrypted with the public key of the exam authority PK_{EA} (Step 8). The exam authority collects the test answer, checks its signature using the candidate's pseudonym, re-signs it, and publishes its encryption with the corresponding candidate's pseudonym as receipt (Step 9).

Marking: the exam authority encrypts the signed test answer with an eligible examiner pseudonym and publishes the encryption on the bulletin board (Step 10). The corresponding examiner marks the test answer, and signs it with his private key (again using the generator h instead of g). The examiner then encrypts it with the exam authority public key, and submits its marks to the exam authority (Step 11).

Notification: when the exam authority receives all the candidate evaluations, it publishes the signed marks, each encrypted with the corresponding candidate's pseudonym (Step 12). Then, the NET de-anonymize the candidate's pseudonyms by revealing its secret exponent r_c (Step 13). Hence the candidate anonymity is revoked, and the mark can finally be registered. Note that, the examiner's secret exponent is not revealed to ensure his anonymity even after the exam concludes.

Formal Analysis. We analyze Remark! protocol with ProVerif, following similar techniques as the one used in the analysis of the H&P protocol. Table 3.2 sums up the results together with the time required for ProVerif to conclude on the same PC used for H&P. We model the bulletin board as a public channel, and use the equational theory depicted in Figure 3.5. The equations for encryption and signatures are standard, but we also added the possibility of using the pseudonym keys to encrypt or sign. The public pseudonym, which also serves as exam-form identifier, is obtained using the function `pseudo_pub` on the public key and the random exponent. The function `pseudo_priv` can be used to decrypt or sign messages, using the private key and the new generator g^r (modeled using the function `exp`) as parameters. The function `checkpseudo` allows us to check if a pseudonym corresponds to a given secret key (or its pseudonym variant).

Candidate Registration:
1- NET calculates $r_c = \prod_{i=1}^{k} r_{e_c}$, $\overline{PK}_C = PK_C^{r_c}$ and $h_c = g^{r_c}$
2- NET publishes on the BB $sign((\overline{PK}_C, h_c), SK_{NET})$
3- C checks if $\overline{PK}_C = h_c^{SK_C}$

Examiner Registration
4- NET calculates $r_e = \prod_{i=1}^{k} r_{e_i}$, $\overline{PK}_E = PK_E^{r_e}$ and $h_e = g^{r_e}$
5- NET publishes on the BB $sign((\overline{PK}_E, h_e), SK_{NET})$
6- E checks if $\overline{PK}_E = h_e^{SK_E}$

Examination:
7- EA publishes on BB $\{sign(question, SK_{EA})\}_{\overline{PK}_C}$
8- $C \rightarrow EA : \{C_a, sign(C_a, \overline{SK_C, h_c})\}_{PK_{EA}}$, where $C_a = \{question, answer, \overline{PK}_C\}$
9- EA publishes on BB $\{C_a, sign(C_a, SK_{EA})\}_{\overline{PK_C}}$

Marking:
10- EA publishes on BB $\{C_a, sign(C_a, SK_{EA})\}_{\overline{PK_E}}$
11- $E \rightarrow EA : \{sign(M_a, \overline{SK_E, h_e})\}_{PK_{EA}}$, where $M_a = (sign(C_a, SK_{EA}), mark)$

Notification:
12- EA publishes on BB $\{M_a, sign(M_a, \overline{SK_E, h_e})\}_{\overline{PK}_C}$
13- $NET \rightarrow EA : \{r_c, sign(r_c, SK_{NET})\}_{PK_{EA}}$

FIGURE 3.4: A symbolic representation of the Remark! protocol.

$$\texttt{checkpseudo}(\texttt{pseudo\_pub}(\texttt{pk}(k), r), \texttt{pseudo\_priv}(k, \texttt{exp}(r))) = \texttt{true}$$
$$\texttt{decrypt}(\texttt{encrypt}(m, \texttt{pk}(k), c), k) = m$$
$$\texttt{decrypt}(\texttt{encrypt}(m, \texttt{pseudo\_pub}(\texttt{pk}(k), r), c), \texttt{pseudo\_priv}(k, \texttt{exp}(r))) = m$$
$$\texttt{getmess}(\texttt{sign}(m, k)) = m$$
$$\texttt{checksign}(\texttt{sign}(m, k), \texttt{pk}(k)) = m$$
$$\texttt{checksign}(\texttt{sign}(m, \texttt{pseudo\_priv}(k, \texttt{exp}(r))), \texttt{pseudo\_pub}(pk(k), r)) = m$$

FIGURE 3.5: Equational theory for our model of Remark! protocol.

Authentication properties. Assuming an attacker in control of the network and all parties to be honest, we can successfully verify all authentication properties in ProVerif. To model properly authentication in ProVerif, where events need to refer to candidates along the whole code, it was necessary to replace the candidate key (used to identify the candidate) with the candidate's pseudonym inside the events. This is sound because there is a bijective mapping between keys and pseudonyms, and pseudonyms are always available. We also verified the authentication properties considering corrupted parties. In this case, all properties are guaranteed except *Form Authenticity*. The attack trace shows that a corrupted candidate can pick the examiner of his 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 will correct his exam. As the protocol description envisages an access control for publishing into the bulletin board, a feature that we did

Property	Result	Time
Answer Origin Authentication	✓ (NET)	< 1 s
Form Authorship	✓ (C, EA, NET)	< 1 s
Form Authenticity	✓ (C, E, EA, NET)	< 1 s
Form Authenticity*	✓ (E, EA, NET)	< 1 s
Mark Authenticity	✓ (E, EA, NET)	< 1 s
Question Indistinguishability	✓ (E, EA, NET)	< 1 s
Anonymous Marking	✓ (C, NET)	1 s
Anonymous Examiner	✓ (E, NET)	< 1 s
Mark Privacy	✓ (EA, NET)	3 m 39 s

TABLE 3.2: Results of our analysis on the formal model of the Remark! protocol. The parties which are assumed to be honest for the result to hold are in brackets. NET is the process that models the mix networks. (*) after applying our fix.

not code in ProVerif, we cannot claim this to be an attack as the candidate may not be allowed to post on the bulletin board. However, we demonstrate that with a simple fix there is no need of access control policies for publishing into the bulletin board. The fix consists in making the intended pseudonym of an examiner explicit within the signature that designates the examiner as evaluator of an exam. In doing so, the exam authority's signature within the receipt cannot be used by a candidate to designate any examiner because the receipt includes no examiner's pseudonym. The exam authority will only accept exam evaluations that contain its signature on examiner's pseudonym. Considering the fix, ProVerif confirms that Remark! guarantees *Form Authenticity*, even in presence of corrupted candidates.

Privacy properties. All the privacy properties are satisfied. For *Question Indistinguishability*, we only assume the exam authority to be honest, and then conclude that the property holds. For *Mark Privacy*, we assume only the concerned candidate and examiner, as well as the exam authority, to be honest. All other candidates and examiners are corrupted, and ProVerif still concludes successfully. Note that, this subsumes a case with multiple honest candidates and examiners, since a dishonest party can behave like a honest party. This also implies that the protocol ensures *Mark Anonymity* as noted above. For *Anonymous Examiner*, we assume only the examiners and the NET to be honest. If the NET publishes the pseudonyms in random order, ProVerif concludes successfully. Similarly for *Anonymous Marking*, we assume only the candidates and the NET to be honest. Again, if the NET publishes the pseudonyms in random order, ProVerif concludes successfully.

3.3.3.3 Protocol of *Université Grenoble Alpes*

The third case study we analyze is Grenoble exam, which is paper-and-pencil procedure used to evaluate undergraduate students at the *Université Grenoble Alpes*.

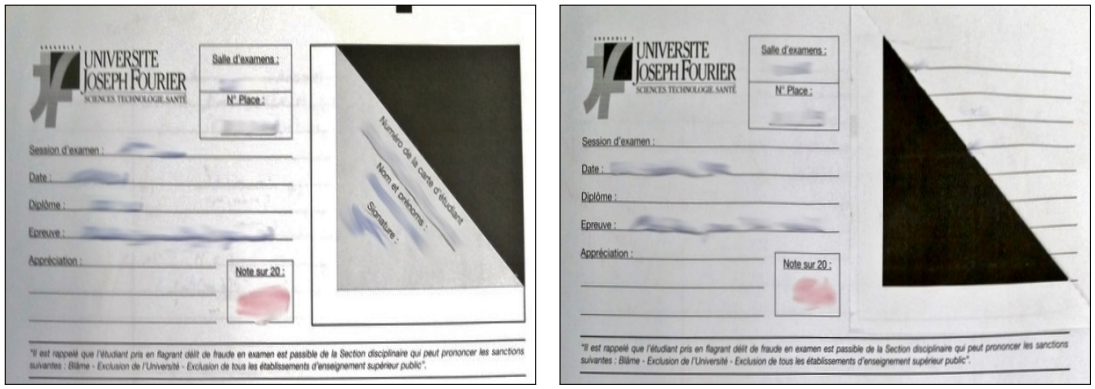


FIGURE 3.6: Special exam paper used in Grenoble exam.

Protocol Description. The protocol involves candidates (C), an examiner (E), a question committee (QC), and an exam authority (EA). It has four phases:

Registration: in Grenoble exam each student has an identity (student name + her birthday), and a pseudonym (student number) which is assigned to her by the exam authority when she registered to the course. All the students of the course are automatically enrolled as eligible candidates for the exam; they are informed about the exam's date, time and location. The QC, the course's lecturer(s), prepares the questions and hands them securely to EA.

Examination: after EA authenticates all candidates, EA lets them take a seat. There, each C finds a special exam paper: the top-right corner is glued and can be folded. Each C signs it, and writes down her name in such a way that the corner, when folded, hides both the signature and the name. Figure 3.6 shows the special paper used in Grenoble exam: a paper with open corner to the left, and a paper with folded corner to the right. Each C also writes down visibly her pseudonym. EA checks that each C writes down her correct name and pseudonym, then the glued part can be folded and sealed. After that, EA distributes the questions and the exam begins. At the end, EA collects the exam-forms, checks that all copies have been returned, that all corners are correctly glued, and gives the exam-forms to E.

Marking: E evaluates the exam-forms: each pseudonym is given a mark. E returns them, along with the marks, to EA.

Notification: for each exam-form, EA checks that the corner is still glued and maps the pseudonym to the real identity without opening the glued part. Then, EA stores the pairs identities/marks, and publishes the pairs pseudonyms/marks. After that, C can review her exam-form in presence of E to check the integrity of her exam-form and verify the mark. If, for instance, C denies that the exam-form containing her pseudonym belongs to her, the glued part is opened.

$$
\begin{aligned}
\texttt{checksign}(\texttt{sign}(m,k),pk(k)) &= m\\
\texttt{getmess}(\texttt{sign}(m,k)) &= m\\
\texttt{unfold}(\texttt{fold}(m,k),k) &= m\\
\texttt{authcheck}(\texttt{auth}(m,s),\texttt{generate}(s)) &= m\\
\texttt{openauth}(\texttt{auth}(m,s)) &= m\\
\texttt{seen}(\texttt{unseen}(m,pk(k),r),k) &= m
\end{aligned}
$$

FIGURE 3.7: Equational theory for our model of Grenoble exam.

Formal Analysis. We analyze Grenoble exam with ProVerif, using the equational theory depicted in Figure 3.7. The obtained results together with the time required for ProVerif to conclude, on the same PC used for the previous case studies, are summarized in Table 3.3. We made a few choices when modeling the Grenoble exam's "visual channels". These are face-to-face channels that all the participants use to exchange data (exam-sheets, student pseudonyms, marks). Intrinsically, all such communications are mutually authenticated. To model visual channels in ProVerif, we could have used private channels, but this would have made the channels too strong, preventing the attacker even from knowing if a communication has happened at all. More appropriately, visual channels are authenticated channels, where authentication is expressed by an equational theory similar to the one commonly used for cryptographic signatures, but with the assumption that the verification key is only known to the intended receiver, namely: $\texttt{openauth}(\texttt{auth}(m,s)) = m$, and $\texttt{authcheck}(\texttt{auth}(m,s),\texttt{generate}(s)) = m$. Function `auth` takes as input a message m and a secret s that only the sender knows, and outputs an authenticated value. The verification key that corresponds to this secret, $\texttt{generate}(s)$, is possessed only by the receiver/verifier. Anyone can get the message, m, but only the owner of $\texttt{generate}(s)$ can verify its origin. The function `fold`, similar to symmetric encryption, is used to hide candidate's identity and signature. The key necessary to reveal the hidden data using the function `unfold` is included inside the message, so that anyone can unfold it. The function `unseen`, similar to asymmetric encryption, is used to model that the attacker cannot see the content of some exchanged messages. For instance when a candidate submits an answer, the others can see that she is submitting an answer but cannot look into its content (this is prevented by the authority which is controlling the exam room). The function `seen` is the inverse of `unseen`. We also use the equational theory of digital signature (functions: `sign`, `checksign` and `getmess`) to model candidate's signature. Note that, we use private channel for the transmission of the questions from QC to EA, as in reality this happen in a secure way (so nobody can see this transmission). Similarly, the authority provides a pseudonym (student number) to the candidate securely. We use a table to model this; EA inserts the identity of the candidate together with her pseudonym in the table, then the candidate gets it. Note

that, in ProVerif, tables cannot be accessed by the attacker. Finally, we assume that an examiner cannot register as a candidate. This is normal since a candidate cannot be an examiner at the same time.

Authentication properties: ProVerif verifies that all the authentication properties are satisfied, if the parties that emits events (necessary for the considered property) are honest. This is necessary for authentication properties, since otherwise the processes may not emit some events when reached.

We make one assumption: the EA only accepts one exam-form per pseudonym. This is realistic as the authority collects only one exam copy from each candidate, which then has to leave the exam room. This assumption is necessary for *Answer Origin Authentication* to hold. Otherwise, the attacker can simply re-submit the candidate's exam-form, and thus the EA will collect twice the same exam-form from the same candidate.

Privacy properties: ProVerif shows that *Question Indistinguishability* is satisfied by Grenoble exam if QC and EA are honest. Otherwise, one of them could reveal the exam questions, and thus break its secrecy. *Anonymous Marking* is satisfied if EA, E, and the two candidates are honest. However, since it is desirable for *Anonymous Marking* to hold even if the examiner is corrupted, we also consider the case where we have a corrupted E. In that case we assume that the examiner still cannot open the glued part (which would trivially break *Anonymous Marking*), as this would be detectable. Given this assumption, ProVerif confirms that *Anonymous Marking* is satisfied by Grenoble exam even if E is corrupted. Concerning *Anonymous Examiner*, ProVerif finds a counterexample even if all parties are honest. The attacker can distinguish which "unseen" exam-form is accepted by the examiner to mark (the one he can "seen" using his secret key). This is not a real attack, since the examiner will only accept exam-forms from the exam authority, not an attacker. If we assume a private channel between the EA and E, ProVerif confirms that *Anonymous Examiner* is satisfied by Grenoble exam even with corrupted candidates. ProVerif finds an attack against *Mark Privacy* (when all parties are honest), this was expected as in Grenoble exam the marks are published in clear-text by the EA. However, *Mark Anonymity* is satisfied in case where we have honest EA, E and two Cs.

3.4 Verifiability in Exams

The authentication properties discussed in the previous section ensure, when satisfied, that only registered candidates can take the exam that no answer can be manipulated, and that each candidate gets the mark attributed to her answers. However, they may require some parties to be honest, *e.g.*, a certain authority, and thus in practice have to be trusted by the other parties, *e.g.*, candidates. In this section we discuss Verifiability properties, which allow parties to check after the end of the exam for the presence or

Property	Result	Time
Answer Origin Authentication	✓(EA)	< 1 s
Form Authorship	✓(C, EA)	< 1 s
Form Authenticity	✓(E, EA)	< 1 s
Mark Authenticity	✓(C, E, EA)	< 1 s
Question Indistinguishability	✓(EA, QC)	< 1 s
Anonymous Marking	✓(C, E, EA)	< 1 s
Anonymous Marking*	✓(C, EA)	< 1 s
Anonymous Examiner	×	< 1 s
Anonymous Examiner†	✓(E, EA)	< 1 s
Mark Privacy	×	< 1 s
Mark Anonymity	✓(C, E, EA)	30 s

TABLE 3.3: Results of our analysis on the formal model of the Grenoble protocol. (*) E corrupted, but cannot open the glued part. (†) private channel between EA and E.

the absence of irregularities and provide evidence about the correctness of the marking procedures. They should be welcome by authorities since exam verifiability is also about to be transparent about an exam being compliant with regulations as well as being able to inspire public trust.

We classify our properties into *individual* and *universal verifiability properties*. The individual properties allow each candidate to verify herself after the end of the exam that none of the answers she submitted were manipulated, that the mark she get was correctly computed on her answers. In their turn verifiability properties allow a generic observer to verify that only registered candidates participated in the exam, that all the accepted (collected) answers were marked without any manipulation, and that all marks were correctly computed and assigned to the corresponding candidates.

We define a new model of exam protocols, which is more general than the previous one. It is inspired by the work of Jannik *et al.* [DJL13]. We propose eleven abstract verifiability properties, then we instantiate them using Applied π-Calculus and we analyze two case studies: the protocols by Giustolisi *et al.* [GLR14], and Grenoble exam. Note that, we do not analyze the verifiability of H&P protocol as it does not provide any verifiability means. We consider the same exam parties: candidates, examiners, question committee, and exam authorities. We also consider the same four phases: Registration, Examination, Marking, and Notification. Assuming such roles and such phases, our model of exam consists of four sets — a set of candidates, a set of questions, a set of answers (questions and answers together are called *exam-forms*) and a set of marks. Three relations link candidates, exam-forms, and marks along the four phases: `Accepted` (corresponds to the event `accept`), `Marked` (corresponds to the event `attribute`), and `Assigned`. They are assumed to be recorded during the exam or build from data logs such as registers or repositories. Note that, the set `Assigned` is different from the event `notify` previously

seen. The former reflects the fact that the authority stores the mark in its database or sends it to the corresponding candidate, while the latter reflects the fact that the candidate receives the mark (in analogues to "submit" and "accept" an answer).

Definition 3.13. (Exam). *An exam* $\mathtt{E}$ *is a tuple* $(\mathtt{I}, \mathtt{Q}, \mathtt{A}, \mathtt{M}, \alpha)$ *where* $\mathtt{I}$ *of type* $\mathbf{I}$ *is a set of candidate identities,* $\mathtt{Q}$ *of type* $\mathbf{Q}$ *is a set of questions,* $\mathtt{A}$ *of type* $\mathbf{A}$ *is a set of answers,* $\mathtt{M}$ *of type* $\mathbf{M}$ *is a set of marks, and* α *is the set of the following relations:*

- $\mathtt{Accepted} \subseteq \mathtt{I} \times (\mathtt{Q} \times \mathtt{A})$*: the candidates' exam-forms accepted by the authority;*
- $\mathtt{Marked} \subseteq \mathtt{I} \times (\mathtt{Q} \times \mathtt{A}) \times \mathtt{M}$*: the marks delivered on the exam-forms;*
- $\mathtt{Assigned} \subseteq \mathtt{I} \times \mathtt{M}$*: the marks assigned (i.e., officially linked) to the candidates;*
- $\mathtt{Correct} : (\mathbf{Q} \times \mathbf{A}) \to \mathbf{M}$*: the function used to mark an exam-form;*

Definition 3.13 is simple but expressive. It can model electronic as well as paper-and-pencil exams, and exams executed honestly as well as exams with frauds. It is the goal of verifiability to test for the absence of anomalies. For this aim we recognize two specific subsets: (a) $\mathtt{I_r} \subseteq \mathtt{I}$ as the set of candidates who registered for the exam (thus, $\mathtt{I} \setminus \mathtt{I_r}$ are the identities of the unregistered candidates who have taken the exam), and (b) $\mathtt{Q_g} \subseteq \mathtt{Q}$ as the questions that the question committee has prepared (thus, $\mathtt{Q} \setminus \mathtt{Q_g}$ are the additional and illegitimate questions that appear in the exam). Note that, set $\mathtt{I_r}$ corresponds to event `register` previously seen. The function `Correct` models any objective mapping that assigns a mark to an answer. This works well for single-choice and with multiple-choice questions. However, it is inappropriate for long open questions because marking an open question is hardly objective: the ambiguities of natural language can lead to subjective interpretations by the examiner. Thus, independently of the model, we cannot hope to verify the marking in such a context. Since in our framework the function `Correct` is used to verify the correctness of the marking, exams that do not allow a definition of such a function cannot be checked for that property; however, all other properties can still be checked.

To be verifiable with respect to specific properties, an exam protocol needs to provide tests to verify these properties. A test t is a function from $\mathcal{E} \to \mathtt{bool}$, where $\mathcal{E}$ is the set of data used to run the test. Abstractly, a verifiable property has the format $t(e) \Leftrightarrow c$, where t is a test, e is the data used, and c is a predicate that expresses the property the test is expected to check. Direction $\Rightarrow$ says that the test's success is a sufficient condition for c to hold (soundness); direction $\Leftarrow$ says that the test's success is a necessary condition for c to hold (completeness).

Definition 3.14. (Verifiable Exam). *An exam for which it exists a test for a property is* testable *for that property. An exam is* verifiable *for that property when it is testable and when the test is* sound *and* complete.

To work, a test needs pieces of data from the exam's execution. A verifier, which is the entity who runs the test, may complementary use personal knowledge about the exam's run if he has any. We assume data to be taken after the exam has ended, that is, when they are stable and not subject to further changes.

To be useful, tests have to be sound even in the presence of an attacker or of dishonest participants: this ensures that when the test succeeds the property holds despite any attempt by the attacker or the participants to falsify it. However, many sound tests are not complete in such conditions: a misbehaving participant can submit incorrect data and, in so doing, causing the test to fail although the property holds. Unless said differently, we check for soundness in presence of some dishonest participants (indeed we seek for the maximal set of dishonest participants that preserve the soundness of the test), but we check for completeness only with honest participants.

A verifiability test can be run by the exam participants or by outsiders. This brings to two distinct notions of verifiability properties: *individual* and *universal*. In exams, individual verifiability means verifiability from the point of view of a candidate. She can feed the test with the knowledge she has about the exam, namely her personal data (identity, exam-form, mark) and the messages she exchanged with the other participants during the exam. Universal verifiability means verifiability from the point of view of an external observer. In practical applications this might be an auditor who has no knowledge of the exam: he has no candidate ID, he has not seen the exam's questions and answered any of them, and he did not receive any mark. Besides, he has not interacted with any of the exam participants. In short, he runs the test only using the exam's public pieces of data available to him.

In the next two sections, we respectively define seven individual and five universal verifiability properties. These properties cover the verifiability of all phases of a typical exam. We define one property about registration verifiability, one about the validity of questions, two about the integrity of exam-test, two about the process of marking, and one about the integrity of notification. More details are given in the remainder of the section. Generally speaking, an exam is fully (individual or universal) verifiable when it satisfies all the properties. Of course, on a particular exam each property can be verified separately to clearly assess its strengths and weaknesses.

3.4.1 Individual Verifiability Properties

Here is the candidate that verifies the exam. She knows her identity i, her submitted exam-form q and a, and her mark m. She also knows her perspective p of the exam run, that is, the messages she has sent and received. Her data is a tuple (i, q, a, m, p). Note that, the candidate's perspective p is not necessary to define the properties, for this reason it does not appear in the right-hand-side of the equivalent. However, it might

be necessary to implement the test depending on the case study. There is no individual verifiability (I.V.) property about registration as a candidate knows whether she has registered, and she might even have a receipt of it. Instead, what a candidate does not know, but wishes to verify, is whether she got the correct questions, and whether she got her form correctly marked.

To verify the validity of her question, we propose the property *Question Validity* which, when satisfied, allows the candidate to check whether the questions she received are actually generated by the question committee. This is modeled by a test which returns true, if and only if, the questions q received by the candidate belong to the set of the valid questions Q_g generated by the question committee.

Definition 3.15. (Question Validity I.V.). *Given an exam* $\mathtt{E}$ *and a set of tests* β*, then* $(\mathtt{E}, \beta)$ *is* question validity verifiable *if there is a test* $\mathtt{QV_{IV}} : \mathcal{E} \rightarrow \mathtt{bool}$ *in* β *s.t.:* $\mathtt{QV_{IV}}(i, q, a, m, p) \Leftrightarrow (q \in Q_g)$

The test $\mathtt{QV_{IV}}$ could be implemented by *e.g.*, checking a signature coming with q, but other implementations are possible. For the candidate to verify that her mark is correct, we propose the property *Marking Correctness* which allows the candidate to check whether the mark she received is correctly computed on her exam-form.

Definition 3.16. (Marking Correctness I.V.). *Let* E *be an exam and* β *be a set of tests, then* (E, β) *is* marking correctness verifiable *if there is a test* $\mathtt{MC_{IV}} : \mathcal{E} \rightarrow \mathtt{bool}$ *in* β *s.t.:* $\mathtt{MC_{IV}}(i, q, a, m, p) \Leftrightarrow (\mathtt{Correct}(q, a) = m)$

Verifying *Marking Correctness* could *e.g.*, be realized by giving access to the marking algorithm, so that the candidate can compute again the mark that corresponds to her exam-form and compare it to the mark she received. This is feasible with multiple-choice questions or short open-questions, but rather difficult in other cases such as the case of long and open questions.

In case that the implementation does not support the verification of *Marking Correctness*, a candidate may wish to verify whether at least the mark she received is the one attributed to her answers. This is ensured by *Correct Mark Reception.*

Definition 3.17. (Correct Mark Reception I.V.). *Let* E *be an exam and* β *be a set of tests, then* (E, β) *is* correct mark reception verifiable *if there is a test* $\mathtt{CMR_{IV}} : \mathcal{E} \rightarrow \mathtt{bool}$ *in* β *s.t.:* $\mathtt{CMR_{IV}}(i, q, a, m, p) \Leftrightarrow ((i, (q, a), m) \in \mathtt{Marked}))$

Correct Mark Reception is sufficient for the candidate to be convinced that she got the correct mark, provided the examiner follows the marking algorithm correctly. Note that, even if *Marking Correctness* is satisfied by an exam, *Correct Mark Reception* may helps when $\mathtt{MC_{IV}}$ fails to identify the source of error. For instance, when $\mathtt{MC_{IV}}$ fails and $\mathtt{CMR_{IV}}$ succeeds, the candidate can be sure that the failure of $\mathtt{MC_{IV}}$ is due to the fact that the

examiner corrected her answer wrongly, and not related to the exam-form integrity before marking or mark integrity after marking.

Again when *Correct Mark Reception* is not satisfied, or if $\texttt{CMR}_{\texttt{IV}}$ fails more properties about candidate's exam-form and mark can be verified in order to grantee some points about them or to identify in which step of the exam the error happened. Precisely whether the integrity of the candidate's exam-form is preserved upon acceptance, whether the accepted exam-form is the one that is marked, and whether the integrity of the candidate's mark is preserved from delivery through assignment until reception. Preserving the integrity of the exam-form and that of the mark is sufficient for the candidate to be convinced that she got the mark delivered on her exam-form. Each these properties covers a different step from exam-form submission until mark reception. The next property *Exam-form Integrity* ensures that the candidate's exam-form is accepted as she submitted it without any modification.

Definition 3.18. (Exam-form Integrity I.V.). *Let E be an exam and β be a set of tests.* (E, β) *is* exam-form integrity verifiable *if there is a test* $\texttt{ETI}_{\texttt{IV}} : \mathcal{E} \rightarrow \texttt{bool}$ *in* β *s.t.:* $\texttt{ETI}_{\texttt{IV}}(i, q, a, m, p) \Leftrightarrow \big((i, (q, a)) \in \texttt{Accepted}\big)$

Another property that also concerns the integrity of the exam-form is *Exam-form Markedness* which ensures that the exam-form accepted from the candidate is marked without modification.

Definition 3.19. (Exam-form Markedness I.V.). *Let E be an exam and β be a set of tests, then* (E, β) *is* exam-form markedness verifiable *if there is a test* $\texttt{ETM}_{\texttt{IV}} : \mathcal{E} \rightarrow \texttt{bool}$ *in* β *s.t.:* $\texttt{ETM}_{\texttt{IV}}(i, q, a, m, p) \Leftrightarrow (\exists!\ (q', a') : (i, (q', a')) \in \texttt{Accepted} \wedge \exists!\ m' : (i, (q', a'), m') \in \texttt{Marked}))$

Running the *Exam-form Markedness* test after the end of the exam does not invalidate the property since if an exam-form is lost or modified before being marked, it remains modified also after the exam is over. But the event consisting of an exam-form that is first changed before the marking, and then restored correctly after marking, is not captured by *Exam-form Markedness*.

The remaining two properties ensure that the integrity of the mark attributed to a candidate by the examiner is preserved until reception. The property *Mark Integrity* ensures that the mark attributed to a candidate is assigned to that candidate by the responsible authority without any modification.

Definition 3.20. (Mark Integrity I.V.). *Let E be an exam and β be a set of tests, then* (E, β) *is* mark integrity verifiable *if there is a test* $\texttt{MI}_{\texttt{IV}} : \mathcal{E} \rightarrow \texttt{bool}$ *in* β *s.t.:* $\texttt{MI}_{\texttt{IV}}(i, q, a, m, p) \Leftrightarrow \exists!\ (q', a') : \big((i, (q', a'), m') \in \texttt{Marked} \wedge \exists!\ m' : (i, m') \in \texttt{Assigned}\big)$.

The last individual property *Mark Notification Integrity* ensures that the candidate receives the mark assigned to her by the authority.

Definition 3.21. (Mark Notification Integrity I.V.). *Let E be an exam and β be a set of tests, then (E, β) is* mark notification integrity verifiable *if there is a test* $\texttt{MNI}_{\texttt{IV}} : \mathcal{E} \to \texttt{bool}$ *in β s.t.:* $\texttt{MNI}_{\texttt{IV}}(i, q, a, m, p) \Leftrightarrow (i, m) \in \texttt{Assigned}$

Relations. If *Correct Mark Reception* is not satisfied, then probable *Exam-form Markedness* and *Mark Integrity* are not satisfied to as this could related to the set `Marked`. However, *Exam-form Integrity* and *Mark Notification Integrity* are independent and may be satisfied. If it is the case, then $\texttt{MI}_{\texttt{IV}}$ and $\texttt{MNI}_{\texttt{IV}}$ guarantee when they succeed that the candidate's exam-form is accepted as submitted, and that the mark she received is the one assigned to her. This allows the candidate to be sure that no problem was encountered during neither answers collection nor mark notification. However, this provides no guarantees about whether the mark she received is actually the correct mark corresponding to his answers, which generally is not satisfying for the candidate.

Now, assuming that all the five properties *Correct Mark Reception*, *Exam-form Integrity*, *Exam-form Markedness*, *Mark Integrity*, *Mark Notification Integrity* are satisfied by an exam. If $\texttt{CMR}_{\texttt{IV}}$ fails, $\texttt{ETI}_{\texttt{IV}}$, $\texttt{ETM}_{\texttt{IV}}$, $\texttt{MI}_{\texttt{IV}}$, $\texttt{MNI}_{\texttt{IV}}$ allows to detect in which step of the exam the error happened. For instance, if $\texttt{CMR}_{\texttt{IV}}$ fails but $\texttt{ETI}_{\texttt{IV}}$ and $\texttt{MNI}_{\texttt{IV}}$ succeeds, then the candidate's exam-form is accepted as submitted, and that the mark she received is the one assigned to her. Thus, the candidate can be sure that an error was happened during marking, that is either her exam-form is modified before marking ($\texttt{ETM}_{\texttt{IV}}$ fails), or she assigned a mark different from the one attribute to her exam-form ($\texttt{MI}_{\texttt{IV}}$ fails).

Note that, the success of $\texttt{ETI}_{\texttt{IV}}$, $\texttt{ETM}_{\texttt{IV}}$ means that the exam-form submitted by the candidate is marked without any modification (Lemma 3.1), and the success of $\texttt{MI}_{\texttt{IV}}$ and $\texttt{MNI}_{\texttt{IV}}$ means that the mark received by the candidate is the one attributed to her by the examiner during marking (Lemma 3.2).

Lemma 3.1. *Providing that* Exam-form Integrity*, and* Exam-form Markedness *are satisfied by a given exam, then*

$$\texttt{ETI}_{\texttt{IV}}(i, q, a, m, p) \wedge \texttt{ETM}_{\texttt{IV}}(i, q, a, m, p) \Rightarrow \exists!\ m' : (i, (q, a), m') \in \texttt{Marked}$$

Proof. We have that, $\texttt{ETI}_{\texttt{IV}}(i, q, a, m, p) \Leftrightarrow (i, (q, a)) \in \texttt{Accepted}$. On the other hand, we have that $\texttt{ETM}_{\texttt{IV}}(i, q, a, m, p) \Leftrightarrow \exists!\ (q', a') : (i, (q', a')) \in \texttt{Accepted} \wedge \exists!\ m' : (i, (q', a'), m') \in \texttt{Marked}$. Then, $(q', a') = (q, a)$ since, $\exists!\ (q', a') : (i, (q', a')) \in \texttt{Accepted}$ and $(i, (q, a)) \in \texttt{Accepted}$. Thus $\exists!\ m' : (i, (q, a), m') \in \texttt{Marked}$. □

Lemma 3.2. *Providing that* Mark Integrity*,* Mark Notification Integrity *are satisfied by a given exam, then*

$$\texttt{MI}_{\texttt{IV}}(i, q, a, m, p) \wedge \texttt{MNI}_{\texttt{IV}}(i, q, a, m, p) \Rightarrow \exists!\ (q', a') : (i, (q', a'), m) \in \texttt{Marked}$$

Proof. We have that, $\texttt{MI}_{\texttt{IV}}(i,q,a,m,p) \Leftrightarrow \exists!\ (q',a') : (i,(q',a'),m') \in \texttt{Marked} \wedge \exists!\ m' : (i,m') \in \texttt{Assigned}$. On the other hand, we have that $\texttt{MNI}_{\texttt{IV}}(i,q,a,m,p) \Leftrightarrow (i,m) \in \texttt{Assigned}$. Then, $m = m'$ since $(i,m) \in \texttt{Assigned}$ and $\exists!\ m' : (i,m') \in \texttt{Assigned}$. Thus, $\exists!\ (q',a') : (i,(q',a'),m) \in \texttt{Marked}$. □

Theorem 3.2. *Providing that* Correct Mark Reception*,* Exam-form Integrity*,* Exam-form Markedness*,* Mark Integrity*,* Mark Notification Integrity *are satisfied by a given exam, then*

$$\texttt{ETI}_{\texttt{IV}}(i,q,a,m,p) \wedge \texttt{ETM}_{\texttt{IV}}(i,q,a,m,p) \wedge \texttt{MI}_{\texttt{IV}}(i,q,a,m,p) \wedge \texttt{MNI}_{\texttt{IV}}(i,q,a,m,p) \Rightarrow \texttt{CMR}_{\texttt{IV}}(i,q,a,m,p)$$

Proof. By Lemma 3.1 we have that $\exists!\ m' : (i,(q,a),m') \in \texttt{Marked}$, and by Lemma 3.2 we have that $\exists!\ (q',a') : (i,(q',a'),m) \in \texttt{Marked}$. Then, we deduce that $(q',a',m') = (q,a,m)$. Thus, $(i,(q,a),m) \in \texttt{Marked}$. Hence, $\texttt{CMR}_{\texttt{IV}}(i,q,a,m,p)$ succeeds, by *Correct Mark Reception.* □

3.4.2 Universal Verifiability Properties

The universal verifiability (U.V.) properties are designed from the viewpoint of a generic observer. In contrast to the individual viewpoint, the observer does not have an identity and does not know an exam-form or a mark, because he does not have an official exam role. The observer runs the test on the public data available after a protocol run. Hence, we simply have a general variable e containing the data.

In the universal perspective, properties such as *Question Validity*, *Correct Mark Reception*, and *Mark Notification Integrity* are not relevant because the external observer has no knowledge of the exam-forms submitted nor of the marks received by the candidates. However, an observer may want to verify other properties revealing whether the exam has been carried out correctly, or he may want to check that the exam authorities and examiners have played by the rules. Precisely, an observer would be interested in verifying that only registered candidates were participated in the exam, all accepted exam-forms are marked without any modification, all exam-forms are marked correctly, and all marks are assigned to the corresponding candidates.

The five universal verifiability properties cover all the exam steps. Thus, an exam system is fully universally verifiable if it guarantees these five properties. However, again each property can be checked independently from the others. The first universal property is *Registration*, which allows to verify that all accepted exam-forms were submitted by registered candidates.

Definition 3.22. (Registration U.V.). *Let E be an exam and β be a set of tests, then (E,β) is* registration verifiable *if there is a test $\texttt{R}_{\texttt{UV}} : \mathcal{E} \to \texttt{bool}$ in β s.t. $\texttt{R}_{\texttt{UV}}(e)) \Leftrightarrow \{i : (i,(q,a)) \in \texttt{Accepted}\} \subseteq I_r$.*

An observer may wish to test that all the marks attributed by the examiners to the exam-forms are computed correctly. This property, *Marking Correctness*, raises the same practical questions as the individual case and therefore the same discussion applies here. However, even in case of open questions, to increase their trustworthiness, universities should allow auditors to access their log for an inspection to the marking process.

Definition 3.23. (Marking Correctness U.V.). *Let E be an exam and β be a set of tests, then (E, β) is* marking correctness universally verifiable *if there is a test* $\texttt{MC}_{\texttt{UV}} : \mathcal{E} \rightarrow \texttt{bool}$ *in β s.t.* $\texttt{MC}_{\texttt{UV}}(e)) \Leftrightarrow (\forall (i, (q, a), m) \in \texttt{Marked},\ \texttt{Correct}(q, a) = m)$.

One may also want to check that no exam-form is modified, added, or deleted until the end of the marking phase. The next property *Exam-form Integrity* allows an observer to verify that all and only accepted exam-forms are marked without any modification.

Definition 3.24. (Exam-form Integrity U.V.). *Let E be an exam and β be a set of tests, then (E, β) is* exam-form integrity universally verifiable *if there is a test* $\texttt{ETI}_{\texttt{UV}} : \mathcal{E} \rightarrow \texttt{bool}$ *in β s.t.* $\texttt{ETI}_{\texttt{UV}}(e) \Leftrightarrow \texttt{Accepted} = \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\}$.

Another property that could be useful when ETI_{UV} fails is *Exam-form Markedness*, which allows us to verify that all the accepted exam-forms are marked without modification.

Definition 3.25. (Exam-form Markedness U.V.). *Let E be an exam and β be a set of tests, then (E, β) is* exam-form markedness universally verifiable *if there exists a test* $\texttt{ETM}_{\texttt{UV}} : \mathcal{E} \rightarrow \texttt{bool}$ *in β s.t.* $\texttt{ETM}_{\texttt{UV}}(e)) \Leftrightarrow \texttt{Accepted} \subset \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\}$.

In case that $\texttt{ETI}_{\texttt{UV}}$ fails but $\texttt{ETM}_{\texttt{UV}}$ succeeds, then there is at least one extra marked exam-form which is not accepted by the exam authority during the examination phase. However, all the accepted exam-forms are marked.

Lemma 3.3. *Providing that Mark Integrity, Mark Notification Integrity are satisfied by a given exam, then* $\texttt{ETI}_{\texttt{UV}}(e) \Rightarrow \texttt{ETM}_{\texttt{UV}}(e)$.

Proof. Suppose that $\texttt{ETI}_{\texttt{UV}}(e)$ succeeds then, by Exam-form Integrity U.V., we have that $\texttt{Accepted} = \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\}$. Thus, $\texttt{Accepted} \subset \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\}$. Hence, by Exam-form Markedness U.V., $\texttt{ETM}_{\texttt{UV}}(e)$ succeeds. □

Finally, an observer may wish to check that all and only the marks assigned to exam-forms are assigned to the corresponding candidates with no modifications. This is guaranteed by *Mark Integrity*.

Definition 3.26. (Mark Integrity U.V.). *Let E be an exam and β a set of tests, then (E, β) is* mark integrity universally verifiable *if there exists a test* $\texttt{MI}_{\texttt{UV}} : \mathcal{E} \rightarrow \texttt{bool}$ *in β s.t.* $\texttt{MI}_{\texttt{UV}}(e) \Leftrightarrow \texttt{Assigned} = \{(i, m) : (i, (q, a), m) \in \texttt{Marked}\}$.

3.4.3 Case Studies

We validate our framework and show its flexibility by analyzing using ProVerif the verifiability properties of two exam protocols: the pencil-and-paper Grenoble exam, and the electronic exam protocol by Giustolisi *et al.* [GLR14].

3.4.3.1 Protocol of *Université Grenoble Alpes*

The first exam that we analyze is the paper-and-pencil procedure used to evaluate undergraduate students at the University of Grenoble. It involves candidates (C), an examiner (E), a question committee (QC), and an exam authority (EA). It has four phases already described in Section 3.3.3.3.

Formal Model. We use the same equational theory already presented in Figure 3.7 to model physical features of Grenoble protocol. Additionally, each data set of Definition 3.13 is composed by a selection of messages taken from the data generated by the processes, possibly manipulated by the attacker. For example, $\mathtt{Q}$ are all the messages that represent a question. $\mathtt{Q_g}$, subset of $\mathtt{Q}$, are all the messages representing a question that are generated by the QC. The exam's relations are also as in Definition 3.13. The set `Accepted` contains all the messages $(i, (q, a))$ (*i.e.*, identity and exam-form) that EA has collected. If the EA is honest, it accepts only the exam-forms submitted by registered candidates. The set `Marked` contains all the messages $(i, (q, a), m)$ (*i.e.*, identity, exam-form, and mark) that the E has generated after having marked the exam-forms. If E is honest, he marks only exam-forms authenticated by EA. The set `Assigned` contains all the messages (i, m) originating from the EA when it assigns mark m to candidate i. If EA is honest, it assigns a mark to C only if E notifies that it is the mark delivered on C's exam-form. The function `Correct` is a deterministic function that outputs a mark for a given exam-form. Note that in ProVerif, we represent the data sets using events inside the process of the parties. For example, an event $\mathtt{accepted}(i, q, a)$ is emitted once the element $(i, (q, a))$ is added to the set `Accepted`. Moreover, the data are sent to the test process on private channels in case of completeness, and on public channels in case of soundness.

Analysis of Individual Verifiability. We model individual verifiability tests as processes in ProVerif. Each test emits two events: the event `OK`, when the test succeeds, and the event `KO`, when the test fails. We use correspondence assertions, *i.e.*, *"if an event e is executed the event e' has been previously executed"* [RSG+00], to prove soundness, and resort to unreachability of `KO` to prove completeness. We also use unreachability to prove soundness for Marking Correctness. A sound test receives its input via public channels. This allows an attacker to mess with the test's inputs. Participants can be dishonest too. Thus, we check that the event `OK` is always preceded by the event emitted in the part of the code where the predicate becomes satisfied. Below, we describe how this works for

Property	Sound	Complete
Question Validity	✓(EA)	✓(all)
Marking Correctness	✓	✓(all)
Correct Mark Reception	✓(EA, E)	✓(all)
Exam-form Integrity	✓(EA)	✓(all)
Exam-form Markedness	✓(EA, E)	✓(all)
Mark Integrity	✓(EA, E)	✓(all)
Mark Notification Integrity	✓(EA)	✓(all)

TABLE 3.4: Results for I.V. properties of the Grenoble exam.

Question Validity. A complete test receives its input via private channels and by honest participants. The attacker cannot change the test's input this time. Then, we check that the test does not fail, that is, the event `KO` is unreachable.

Table 3.4 reports the result of the analysis. All properties hold (✓) despite the attacker, but often they hold only assuming some roles to be honest, otherwise attacks exist. All properties but Marking Correctness have sound tests (Table 3.4, middle column) only if we assume at least the honesty of the exam authority (EA), or of the examiner (E), or of both. This in addition to the honesty of candidate, who must be necessarily honest because he is the verifier. The minimal assumptions for all the properties are reported in brackets. All properties have complete tests (Table, right columns) but all roles except the attacker have to be honest for them to hold. In the following we describe the tests used to verify the individual properties of Grenoble exam.

Question Validity: Figure 3.8 presents the code for the Question Validity's test. The QV test inputs the verification value `ver_AC`, which is used to authenticate the exam authority. On channel `chTest`, the test inputs the authenticated question `auth_q`, which it checks for origin-authenticity. The test succeeds if the question is authenticated by the EA, it fails otherwise. The test emits the event `OK` when it succeeds, otherwise emits the event `KO`. In the proof for soundness, we modified the ProVerif code for EA in such way

```
let test(chTest, ver_AC) =
in(chTest, (auth_q));
let question = openauth(auth_q) in
if authcheck(auth_q, Ver_AC) = question then
event OK
else
event KO.
```

FIGURE 3.8: The Question Validity test for the Grenoble exam.

to emit an event `valid` just after the process receives the question from QC and checks the authenticity of its origin, and just before EA sends the question to the C. ProVerif shows, in case of honest EA, that any `OK` is preceded by `valid`: the test outputs true only

if the question is generated by QC. Note that any tampering that QC can perform on the questions (for example, generating dummy questions or by trashing them after having generated them) does not violate question validity *per se*: according to this property the questions that C received are still those generated, honestly or dishonestly, by the QC: the origin of the question is not compromised. In the proof for completeness, ProVerif shows that the event `KO` is unreachable when all participants are honest.

Marking Correctness: in Grenoble exam when the candidate examine her marked exam-form in presence of the examiner, she can check with the examiner if the marking was done correctly. Then, the Marking Correctness test for Grenoble exam simply takes the exam-form submitted by the candidate and the mark she received, and then compare if the mark obtained by running the marking algorithm on the submitted exam-form is equal to the received mark. The Marking Correctness test is presented in Figure 3.9. Using ProVerif we show that the Marking Correctness test is complete and sound even if all participants are dishonest.

```
let test(chCand) =
in(chCand, (ques, ans, mark));
if mark = correction(ques, ans) then
   event OK;
   if mark <> correction(ques, ans) then event reject
   else 0
else
event KO.
```

FIGURE 3.9: The Marking Correctness test for Grenoble exam.

Correct Mark Reception: the candidate can examine her exam-form after marking and check whether it is marked. So, whether the submitted exam-form is equal to the marked one, and whether the received mark is equal to the one attributed to the exam-form. We show using ProVerif that Correct Mark Reception is complete and sound in case of honest examiner for Grenoble exam.

Exam-form Integrity: the candidate can check the integrity of her exam-form by comparing the the exam-form she submitted to the one marked by the examiner, as she can consult the later after marking. If the exam-form is unmodified after marking, then it is supposed that it is accepted correctly by the exam authority and not modified before marking. The Exam-form Integrity test is presented in Figure 3.11. The test takes from the candidate, on the channel `chCand` (private for completeness and public for soundness), the form she submitted (hidden identity, pseudonym, question, answer and hidden signature). Then it takes the form accepted by the exam authority `auth_fA` from that candidate, and verifies that it is equal to the form submitted by the candidate *i.e.*, the exam authority

```
let test(chCand, chA,  chM,  ver_t) =
in(chCand, (hC, pseuC, ques, ans, hS, m));
in(chM,  auth_fM);
let (=hC, =pseuC, quesX, ansX, hSX, mX) = openauth(auth_fM) in
if (hC, pseuC, quesX, ansX, hSX, mX) = authcheck(auth_fM, ver_t)
then
if ques = quesX && ans = ansX && m = mX then
event OK
else
event KO.
```

FIGURE 3.10: The Correct Mark Reception test for Grenoble exam.

recorded the submitted form without any modification. If its the case the test outputs true (`OK`), and false (`KO`) otherwise. Note that, the test checks the authenticity of the forms received from the exam authority with the verification value `ver_t` since in practice the candidate takes the forms from them by hand. Also, in case of completeness the channel `chA` which used to take the forms from the exam authority is a private channel, while it is public (controlled by the attacker) in case of soundness. ProVerif proves that the Exam-form Integrity test is complete and sound in case of honest examiner and honest exam authority for Grenoble exam.

```
let test(chCand, chA,  chM,  ver_t) =
in(chCand, (hC, pseuC, ques, ans, hS));
in(chA,  auth_fA);
let (=hC, =pseuC, quesX, ansX, hSX) = openauth( auth_fA) in
if (hC, pseuC, quesX, ansX, hSX) = authcheck( auth_fA, ver_t) then
if quesX = ques && ansX = ans then
event OK
else
event KO.
```

FIGURE 3.11: The Exam-form Integrity test for Grenoble exam.

Exam-form Markedness: similar to Exam-form Integrity, the candidate can examine her exam-form after marking and check whether it is marked. So, the exam-form markedness test for Grenoble exam is exactly similar to the exam-form integrity test except that for exam-form markedness the test additionally takes the form marked by the examiner on channel `chM`. Thus, the test checks that if the accepted exam-form is equal to the marked one. We show using ProVerif that Exam-form Markedness is complete and sound in case of honest examiner for Grenoble exam.

Mark Integrity: as the candidate can examine her exam-form after marking, so she can

```
let test(chCand, chA,  chM,  ver_t) =
in(chCand, (hC, pseuC, ques, ans, hS));
in(chA,  auth_fA);
let (=hC, =pseuC, quesX, ansX, hSX) = openauth( auth_fA) in
if (hC, pseuC, quesX, ansX, hSX) = authcheck( auth_fA, ver_t) then
in(chM,  auth_fM);
let (=hC, =pseuC, quesX', ansX', hSX', m) = openauth(auth_fM) in
if (hC, pseuC, quesX', ansX', hSX', m) = authcheck(auth_fM, ver_t)
then
if quesX' = quesX && ansX' = ansX then
event OK
else
event KO.
```

FIGURE 3.12: The Exam-form Markedness test for Grenoble exam.

learn the mark attributed by the examiner for her. The candidate also can asks the exam authority about the mark assigned for her. Thus, a simple test exists for Mark Integrity that is comparing the two marks. The test is presented in Figure 3.13. ProVerif shows that Grenoble exam ensures the completeness and soundness, in case of honest exam authority and examiner, of the Mark Integrity test.

```
let test(chCand, chA, chM,  ver_t) =
in(chCand, (hC, pseuC, ques, ans, hS));
in(chM,  auth_fM);
let (=hC, =pseuC, quesX, ansX, hSX, m) = openauth(auth_fM) in
if (hC, pseuC, ques, ans, hS, m)  = authcheck(auth_fM, ver_t) then
in(chA,  auth_mA);
let (=hC, =pseuC,  mX) = openauth(auth_mA) in
if  (hC, pseuC, mX)  = authcheck(auth_mA, ver_t) then
if mX = m then
event OK
else
event KO.
```

FIGURE 3.13: The Mark Integrity test for Grenoble exam.

Mark Notification Integrity: the candidate can ask the exam authority about the mark assigned to her and check if it is the same one she received. The Mark Notification Integrity test is presented in Figure 3.14. ProVerif shows that the test is complete and sound in case of honest exam authority.

Analysis of Universal Verifiability. Universal verifiable tests should use some public data. However, since the Grenoble exam is a paper-and-pencil based exam, in general, there is no publicly available data. Thus, originally Grenoble exam does not satisfy any of the universal verifiability properties. To be universally testable, an auditor has to

```
let test(chCand:channel, chA:channel,  ver_t:public) =
in(chCand, (hC, pseuC, m));
in(chA,  auth_mA);
let (=hC, =pseuC,  mX) = openauth(auth_mA) in
if  (hC, pseuC, mX)  = authcheck(auth_mA, ver_t) then
if  mX = m then
event OK
else
event KO.
```

FIGURE 3.14: The Mark Notification Integrity test for Grenoble exam.

Property	Sound	Complete
Registration	✓(EA)	✓(all)
Exam-form Integrity	✓(EA, E)	✓(all)
Exam-form Markedness	✓(EA, E)	✓(all)
Marking Correctness	✓(E)	✓(all)
Mark Integrity	✓(EA, E)	✓(all)

TABLE 3.5: Results for U.V. properties of the Grenoble exam.

be given access to the following data: (1) for Registration verifiability, he can read the list of registered candidates and the set of accepted exam-forms. Thus, he can check whether all accepted exam-forms are submitted by registered candidates; (2) for Exam-form Markedness, in addition to the accepted exam-forms, he knows the set of marked exam-forms. Then, he can check whether all the accepted exam-forms are marked; (3) for Exam-form Integrity, he knows the same data as in Exam-form Markedness. The auditor has to check that all and only the accepted exam-forms are marked; (4) for Marking Correctness, he knows the correction algorithm and the marked exam-forms together with the delivered marks. The test is to run the correction algorithm again on each exam-form and check if the obtained mark is the same as the delivered one; finally, for (5) Mark Integrity, in addition to the delivered marks, he can access the assigned marks. The auditor can check whether the assigned marks are exactly the ones delivered and whether they are assigned to the correct candidates. Having access to such significant data mentioned above could break candidate's privacy (for instance identities, answers, and marks can be disclosed to the auditor); that noticed, discussing the compatibility between the universal verifiability and privacy is not in the scope of this thesis. Similar to what we did for the individual verifiability tests, we use correspondence assertions to prove soundness and unreachability of a `KO` event to prove completeness.

Table 3.5 depicts the result of the analysis. We must note that in our testing universal verifiability, we were not able to run for all tests a fully automatically analysis in the general case requiring any number of participants. This is because ProVerif does not support loops and to prove the general case we would have needed to iterate over *e.g.*,

all candidates. For these tests we ran ProVerif only for the base case, that where we have only one accepted exam-form or one assigned mark; then we completed a manual induction proof that generalizes this result to the general case with an arbitrary number of candidates. In the following, we present the universal tests used for Grenoble exam, together with the corresponding manual proofs.

Registration: the Registration test for Grenoble exam is presented in Figure 3.15. We show, with ProVerif, that Registration test is complete and sound, with honest exam authority, for the case where we have one accepted exam-form and any number of registered candidates. Then, we generalize it to the case of any number of accepted exam-forms, more precisely, we show that

$$\mathtt{RV}(E) = \mathtt{true} \Leftrightarrow \mathtt{i} : (\mathtt{i}, (\mathtt{q}, \mathtt{a})) \in \mathtt{Accepted} \subseteq I_r$$

holds for any size n of the set `Accepted` and any number m of registered candidates. Let $\mathtt{RV}_\mathtt{k}(\cdot)$ denotes the Registration test that can be applied to an exam execution that has k accepted exam-forms, and $\mathtt{RV}_\mathtt{k}(\cdot) \rightarrow^* \mathtt{OK}$ denotes that the test outputs `OK` (true) after some steps given certain exam execution with k accepted exam-forms. Let E be an exam execution that has m registered candidates and n accepted exam-forms, and let E_j denotes a version of E where only the j^{th} accepted exam-form is counted, and assume that it is submitted by the candidate i_j. We show with ProVerif that the test is complete and sound for one accepted exam-form and any number of registered candidates, thus we have for soundness

$$\forall 1 \leq j \leq n : \mathtt{RV}_1(E_j) \rightarrow^* \mathtt{OK} \Rightarrow i_j \in I_r$$

and for completeness

$$\forall 1 \leq j \leq n : i_j \in I_r \Rightarrow \mathtt{RV}_1(E_j) \rightarrow^* \mathtt{OK}.$$

The test $\mathtt{RV}_n(E)$ checks if each of the accepted exam-forms received on the channels `chA1`, ..., `chAn` is submitted by one of the registered candidates given on the channels `chR1`, ..., `chRn`, and $\forall 1 \leq j \leq n : \mathtt{RV}_1(E_j)$ checks if the j^{th} accepted exam-form received on the channel `chAj` is submitted by the one of the registered candidates given on the channels `chR1`, ..., `chRn`. Thus, for soundness, we have

$$\begin{gathered}
\mathtt{RV}_n(E) \rightarrow^* \mathtt{OK} \\
\Downarrow \\
\forall 1 \leq j \leq n : \mathtt{RV}_1(E_j) \rightarrow^* \mathtt{OK} \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : i_j \in I_r \\
\Downarrow
\end{gathered}$$

$$\mathtt{i} : (\mathtt{i}, (\mathtt{q}, \mathtt{a})) \in \mathtt{Accepted} \subseteq I_r$$

We can make a similar argument for completeness:

$$\begin{gathered}
\mathtt{i} : (\mathtt{i}, (\mathtt{q}, \mathtt{a})) \in \mathtt{Accepted} \subseteq I_r \\
\Downarrow \\
\forall 1 \leq j \leq n : i_j \in I_r \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : \mathtt{RV}_1(E_j) \rightarrow^* \mathtt{OK} \\
\Downarrow \\
\mathtt{RV}_n(E) \rightarrow^* \mathtt{OK}
\end{gathered}$$

```
let testRV_n(chR1, ..., chRm, chA1, ..., chAn, ver_t) =
in(chR1,  auth_c1); ... in(chRm,  auth_cn);
let (pkC1, pseuC1) = openauth(auth_c1) in
...
let (pkCn, pseuCm) = openauth(auth_cm) in
if (pkC1, pseuC1) = authcheck(auth_c1, ver_t)
    && ... &&
   (pkCn, pseuCm) = authcheck(auth_cm, ver_t)
then
in(chA1,  auth_x1); ... in(chAn,  auth_xn);
let (pkX1, pseuX1, ques1, ans1, hS1) = openauth(auth_x1) in
...
let (pkXn, pseuXn, quesn, ansn, hSn) = openauth(auth_xn) in
if (pkX1, pseuX1, ques1, ans1, hS1) = authcheck(auth_x1, ver_t)
    && ... &&
   (pkXn, pseuXn, quesn, ansn, hSn) = authcheck(auth_xn, ver_t)
then
if (pkX1 = pkC1 && pseuX1 = pseuC1)
   || ... ||
   (pkX1 = pkCm && pseuX1 = pseuCm)
   && ... &&
   (pkXn = pkC1 && pseuXn = pseuC1)
   || ... ||
   (pkXn = pkCm && pseuXn = pseuCm)
then
event OK
else
event KO.
```

FIGURE 3.15: The universal Registration test for Grenoble exam.

Exam-form Integrity: the Exam-form Integrity test checks whether all and only the accepted exam-forms are marked. The Exam-form Integrity test for Grenoble exam is presented in Figure 3.16. We show, with ProVerif, that Exam-form Integrity test is complete and sound, with honest exam authority and examiner, for the case of one

accepted exam-form and one marked exam-form. Then, we generalize it to the case of any number of accepted exam-forms, more precisely, we show that

$$\texttt{ETI}(E) = \texttt{true} \Leftrightarrow \texttt{Accepted} = (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked}$$

holds for any size of the sets `Accepted` and `Marked`. Without loss of generality, we assume that the size of the set `Accepted` is equal to that of `Marked`, as the test can be preceded by a simple check on the sizes of the two sets to see if they are equal or not. If they have different sizes then this means that one of the accepted exam-form is not marked or the reverse, and in such a case the test can outputs false directly. Let $\texttt{ETI}_\texttt{k}(\cdot)$ denotes the Exam-form Integrity test that can be applied to an exam execution that has k accepted exam-forms and k marked exam-form, and let $\texttt{ETI}_\texttt{k}(\cdot) \rightarrow^* \texttt{OK}$ denotes that the test outputs `OK` after some steps for a given exam execution with k accepted and k marked exam-forms. Let E be an exam execution that has n accepted exam-forms and n marked exam-forms, and let E_j denotes a version of E where only the j^{th} accepted exam-form (q_j, a_j), which is submitted by the candidate i_j, and j^{th} marked exam-form (q'_j, a'_j), which is related to the candidate i'_j, are counted. Note that we assume that the exam-forms are marked in the same order as they were accepted. We show with ProVerif that the test is complete and sound for one exam-form, we have for soundness

$$\forall 1 \leq j \leq n : \texttt{ETI}_1(E_j) \rightarrow^* \texttt{OK} \Rightarrow (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j))$$

and for completeness

$$\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j)) \Rightarrow \texttt{ETI}_1(E_j) \rightarrow^* \texttt{OK}.$$

The $\texttt{ETI}_n(E)$ checks if all the accepted exam-forms received on the channels `chA1`, ..., `chAn` are exactly the marked exam-forms received on channels `chM1`, ..., `chMn`, and $\forall 1 \leq j \leq n : \texttt{ETI}_1(E_j)$ checks if the j^{th} accepted exam-form on the channel `chAj` is exactly the exam-form marked on channel `chMj`. Thus, for soundness, we have

$$\begin{gathered}
\texttt{ETI}_n(E) \rightarrow^* \texttt{OK} \\
\Downarrow \\
\forall 1 \leq j \leq n : \texttt{ETI}_1(E_j) \rightarrow^* \texttt{OK} \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j)) \\
\Downarrow \\
\texttt{Accepted} = (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked}
\end{gathered}$$

We can make a similar argument for completeness:

$$\texttt{Accepted} = (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked}$$

$$\Downarrow$$
$$\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j))$$
$$\Downarrow$$
$$\forall 1 \leq j \leq n : \texttt{ETI}_1(E_j) \rightarrow^* \texttt{OK}$$
$$\Downarrow$$
$$\texttt{ETI}_n(E) \rightarrow^* \texttt{OK}$$

```
let testETI_n(chM1, ... , chMn, chA1, ... , chAn, ver_t) =
in(chA1,  auth_fA1); ... in(chAn,  auth_fAn);
let (hC1, pseuC1, ques1, ans1, hS1) = openauth(auth_fA1) in
...
let (hCn, pseuCn, quesn, ansn, hSn) = openauth(auth_fAn) in
if (hC1, pseuC1, ques1, ans1, hS1) = authcheck(auth_fA1, ver_t)
   && ... &&
   (hCn, pseuCn, quesn, ansn, hSn) = authcheck(auth_fAn, ver_t)
then
in(chM1, auth_fM1); ... in(chMn, auth_fMn);
let (hX1, pseuX1, quesX1, ansX1, hSX1, m1) = openauth(auth_fM1) in
...
let (hXn, pseuXn, quesXn, ansXn, hSXn, mn) = openauth(auth_fMn) in
if (hX1, pseuX1, quesX1, ansX1, hSX1, m1) = authcheck(auth_fM1, ver_t)
   && ... &&
   (hXn, pseuXn, quesXn, ansXn, hSXn, mn) = authcheck(auth_fMn, ver_t)
then
if (hX1=hC1 && pseuX1=pseuC1 && quesX1=ques1 && ansX1=ans1 && hSX1=hS1)
   && ... &&
   (hXn=hCn && pseuXn=pseuCn && quesXn=quesn && ansXn=ansn && hSXn=hSn)
then
event OK
else
event KO.
```

FIGURE 3.16: The universal Exam-form Integrity test for Grenoble exam.

Exam-form Markedness: we assume that a third party can takes the marked exam-forms from the examiner, so that he can checks whether all the accepted exam-forms (which he can takes from exam authority) are marked. The Exam-form Markedness test for Grenoble exam is presented in Figure 3.17. We show, with ProVerif, that Exam-form Markedness test is complete and sound, with honest exam authority and examiner, for the case where we have one accepted exam-form and any number of marked exam-forms. Then, we generalize it to the case of any number of accepted exam-forms, more precisely, we show that

$$\texttt{ETM}(E) = \texttt{true} \Leftrightarrow \texttt{Accepted} \subseteq (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked}$$

holds for any size n of the set `Accepted` and any number m of marked exam-forms. Let $\texttt{ETM}_\texttt{k}(\cdot)$ denotes the Exam-form Markedness test that can be applied to an exam execution that has k accepted exam-forms, and $\texttt{ETM}_\texttt{k}(\cdot) \rightarrow^* \texttt{OK}$ denotes that the test outputs `OK` after some steps given certain exam execution with k accepted exam-forms. Let E be an exam execution that has n accepted exam-forms and m marked exam-forms, and let E_j denotes a version of E where only the j^{th} accepted exam-form (q_j, a_j) is counted,

and assume that it is submitted by the candidate i_j. As we show with ProVerif that the test is complete and sound for one accepted exam-form and any number of marked exam-forms, thus we have for soundness

$$\forall 1 \leq j \leq n : \texttt{ETM}_1(E_j) \rightarrow^* \texttt{OK} \Rightarrow (i_j, (q_j, a_j)) \subseteq (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked}$$

and for completeness

$$\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) \subseteq (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked} \Rightarrow \texttt{ETM}_1(E_j) \rightarrow^* \texttt{OK}.$$

The test $\texttt{ETM}_n(E)$ checks if each of the accepted exam-forms received on the channels `chA1`, ..., `chAn` is one of the marked exam-forms received on the channels `chM1`, ..., `chMm`, and $\forall 1 \leq j \leq n : \texttt{ETM}_1(E_j)$ checks if the j^{th} accepted exam-form received on the channel `chAj` is one of the marked exam-forms received on the channels `chM1`, ..., `chMm`. Thus, for soundness, we have

$$\begin{gathered}
\texttt{ETM}_n(E) \rightarrow^* \texttt{OK} \\
\Downarrow \\
\forall 1 \leq j \leq n : \texttt{ETM}_1(E_j) \rightarrow^* \texttt{OK} \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) \subseteq (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked} \\
\Downarrow \\
\texttt{Accepted} \subseteq (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked}
\end{gathered}$$

We can make a similar argument for completeness:

$$\begin{gathered}
\texttt{Accepted} \subseteq (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked} \\
\Downarrow \\
\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) \subseteq (\texttt{i}, (\texttt{q}, \texttt{a})) : (\texttt{i}, (\texttt{q}, \texttt{a}), \texttt{m}) \in \texttt{Marked} \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : \texttt{ETM}_1(E_j) \rightarrow^* \texttt{OK} \\
\Downarrow \\
\texttt{ETM}_n(E) \rightarrow^* \texttt{OK}
\end{gathered}$$

Marking Correctness: we assume that anyone can access the correction algorithm, so a third party can check whether the delivered marks are computed correctly provided he given an access to the marked exam-forms also. The Marking Correctness test is presented in Figure 3.18. Using ProVerif, we show that the Marking Correctness test complete and sound, with honest examiner, in the case where we have only one marked exam-form, *i.e.*, size of `Marked` is 1. Then, we generalize it to the case of any number of

```
let testETM_n(chM1, ..., chMm, chA1, ..., chAn, ver_t) =
in(chA1,  auth_fA1); ... in(chAn,  auth_fAn);
let (hC1, pseuC1, ques1, ans1, hS1) = openauth(auth_fA1) in
...
let (hCn, pseuCn, quesn, ansn, hSn) = openauth(auth_fAn) in
if  (hC1, pseuC1, ques1, ans1, hS1) = authcheck(auth_fA1, ver_t)
    && ... &&
    (hCn, pseuCn, quesn, ansn, hSn) = authcheck(auth_fAn, ver_t)
then
in(chM1,  auth_fM1); ... in(chMm,  auth_fMn);
let (pkX1, pseuX1, quesX1, ansX1, hSX1, mark1) = openauth(auth_fM1) in
...
let (pkXn, pseuXn, quesXn, ansXn, hSXm, markm) = openauth(auth_fMm) in
if (pkX1, pseuX1, quesX1, ansX1, hSX1, mark1) = authcheck(auth_x1, ver_t)
   && ... &&
   (pkXn, pseuXn, quesXn, ansXn, hSXm, markm)= authcheck(auth_xn, ver_t)
then
if (pkC1 = pkX1 && pseuC1 = pseuX1 && ques1 = quesX1 && ans1 = ansX1)
   || ... ||
   (pkC1 = pkXm && pseuC1 = pseuXm && ques1 = quesXm && ans1 = ansXm)
   && ... &&
   (pkCn = pkX1 && pseuCn = pseuX1 && quesn = quesX1 && ansn = ansX1)
   || ... ||
   (pkCn = pkXm && pseuCn = pseuXm && quesn = quesXm && ansn = ansXm)
then
event OK
else
event KO.
```

FIGURE 3.17: The universal Exam-form Markedness for Grenoble exam.

marked exam-forms, more precisely, we show that

$$\texttt{MC}(E) = \texttt{true} \Leftrightarrow \forall (i, (q, a), m) \in \texttt{Marked},\ \texttt{Correct}(q, a) = m$$

holds for any size n of the set `Marked`. Let $\texttt{MC}_\texttt{k}(\cdot)$ denotes the Marking Correctness test that can be applied to an exam execution that has k marked exam-forms, and $\texttt{MC}_\texttt{k}(\cdot) \rightarrow^* \texttt{OK}$ denotes that the test outputs `OK` after some steps for a given exam execution with k marked exam-forms. Let E be an exam execution that has n marked exam-forms, and let E_j denotes a version of E where only the j^{th} marked exam-form (q_j, a_j) is counted, which is submitted by the candidate i_j and attributed the mark m_j *i.e.*, $(i_j, (q_j, a_j), m_j) \in \texttt{Marked}$. As we show with ProVerif that the test is complete and sound for one marked exam-form, we have for soundness

$$\forall 1 \leq j \leq n : \texttt{MC}_1(E_j) \rightarrow^* \texttt{OK} \Rightarrow \texttt{Correct}(q_j, a_j) = m_j$$

and for completeness

$$\forall 1 \leq j \leq n : \texttt{Correct}(q_j, a_j) = m_j \Rightarrow \texttt{MC}_1(E_j) \rightarrow^* \texttt{OK}.$$

The $\texttt{MC}_n(E)$ checks if all the marked exam-forms received on the channels `chM1`, ..., `chMn`

```
let testMC_n(chM1, ..., chMn,  ver_t) =
in(chM1,  auth_fM1); ... in(chMn,  auth_fMn);
let (hC1, pseuC1, ques1, ans1, hS1, m1) = openauth(auth_fM1) in
...
let (hCn, pseuCn, quesn, ansn, hSn, mn) = openauth(auth_fMn) in
if (hC1, pseuC1, ques1, ans1, hS1, m1) = authcheck(auth_fM1, ver_t)
   && ... &&
   (hCn, pseuCn, quesn, ansn, hSn, mn) = authcheck(auth_fMn, ver_t)
then
if  m1 = correction(ques1, ans1)
&&
...
&&
mn = correction(quesn, ansn)
then
event OK
else
event KO.
```

FIGURE 3.18: The universal Marking Correctness test for Grenoble exam.

are marked correctly, and $\forall 1 \leq j \leq n : \mathtt{MC}_1(E_j)$ checks if the j^{th} marked exam-form received on the channel `chMj` is marked correctly. Thus, we have for soundness,

$$\begin{gathered}
\mathtt{MC}_n(E) \rightarrow^* \mathtt{OK} \\
\Downarrow \\
\forall 1 \leq j \leq n : \mathtt{MC}_1(E_j) \rightarrow^* \mathtt{OK} \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : \mathtt{Correct}(q_j, a_j) = m_j \\
\Downarrow \\
\forall (i, (q, a), m) \in \mathtt{Marked},\ \mathtt{Correct}(q, a) = m
\end{gathered}$$

We can make a similar argument for completeness:

$$\begin{gathered}
\forall (i, (q, a), m) \in \mathtt{Marked},\ \mathtt{Correct}(q, a) = m \\
\Downarrow \\
\forall 1 \leq j \leq n : \mathtt{Correct}(q_j, a_j) = m_j \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : \mathtt{MC}_1(E_j) \rightarrow^* \mathtt{OK} \\
\Downarrow \\
\mathtt{MC}_n(E) \rightarrow^* \mathtt{OK}
\end{gathered}$$

Mark Integrity: we assume that a third party can take the list of assigned mark with the corresponding candidates from the exam authority. Thus, simply he can check whether

the assigned marks are exactly the delivered ones and that they assigned to the correct candidates. The Mark Integrity test is presented in Figure 3.19. We show, with ProVerif,

```
let testMI_n(chM1, .. , chMn, chA1, ... , chAn, ver_t) =
in(chM1,  auth_fM1); ... in(chMn,  auth_fMn);
let (hC1, pseuC1, ques1, ans1, hS1, m1) = openauth(auth_fM1) in
...
let (hCn, pseuCn, quesn, ansn, hSn, mn) = openauth(auth_fMn) in
if (hC1, pseuC1, ques1, ans1, hS1, m1) = authcheck(auth_fM1, ver_t)
    && ... &&
    (hCn, pseuCn, quesn, ansn, hSn, mn) = authcheck(auth_fMn, ver_t)
then
in(chA1,  auth_A1); ... in(chAn,  auth_An);
let (hX1, pseuX1, mX1) = openauth(auth_A1) in
...
let (hXn, pseuXn, mXn) = openauth(auth_An) in
if (hX1, pseuX1, mX1) = authcheck(auth_A1, ver_t)
   && ... &&
   (hXn, pseuXn, mXn) = authcheck(auth_An, ver_t)
then
if (hX1 = hC1 && pseuX1 = pseuC1 && mX1 = m1)
   && ... &&
  (hXn = hCn && pseuXn = pseuCn && mXn = mn)
then
event OK
else
event KO.
```

FIGURE 3.19: The universal Mark Integrity test for Grenoble exam.

that Mark Integrity test is complete and sound, with honest exam authority and examiner, in the case where we have only one delivered mark and only one assigned mark. Then, we generalize it to the case of any number of delivered and assigned marks, more precisely, we show that

$$\texttt{MI}(E) = \texttt{true} \Leftrightarrow \texttt{Assigned} = (\texttt{i},\texttt{m}) : (\texttt{i},(\texttt{q},\texttt{a}),\texttt{m}) \in \texttt{Marked}$$

holds for any size of the sets `Marked` and `Assigned`. Without loss of generality, we assume that the size of the set `Assigned` is equal to that of `Marked`, as the test can be preceded by a simple check to see whether the sizes of the two sets are equal or not. If they have different sizes then this means that one of the delivered marks is not assigned to any of the candidates or a candidate is assigned a mark which is not delivered, and in such a case the test can output false directly. Let $\texttt{MI}_\texttt{k}(\cdot)$ denotes the Mark Integrity test that can be applied to an exam execution that has k delivered marks and k assigned marks, and let $\texttt{MI}_\texttt{k}(\cdot) \rightarrow^* \texttt{OK}$ denotes that the test outputs `OK` after some steps for a given exam execution with k delivered and k assigned marks. Let E be an exam execution that has

n delivered marks and n assigned marks, and let E_j denotes a version of E where only the j^{th} mark m_j delivered for i_j, and the j^{th} mark m'_j assigned to j'_j are counted. Note that, we assume that the assigned marks are ordered in a table as they were delivered. As we show with ProVerif that the test is complete and sound for one exam-form, we have for soundness

$$\forall 1 \leq j \leq n : \mathtt{MI}_1(E_j) \rightarrow^* \mathtt{OK} \Rightarrow (i_j, m_j) = (i'_j, m'_j)$$

and for completeness

$$\forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \Rightarrow \mathtt{MI}_1(E_j) \rightarrow^* \mathtt{OK}.$$

The $\mathtt{MI}_n(E)$ checks if all the delivered marks received on the channels `chM1`, ..., `chMn` are equal to the assigned marks received on the channels `chA1`, ..., `chAn` and that they are assigned to the correct candidates *i.e.*, they are assigned correctly without modification, and $\forall 1 \leq j \leq n : \mathtt{MI}_1(E_j)$ checks if the j^{th} delivered mark received on the channel `chMj` is equal to the assigned mark received on the channel `chAj` and that it is assigned to the correct candidate. Thus, for soundness, we have

$$\begin{gathered}
\mathtt{MI}_n(E) \rightarrow^* \mathtt{OK} \\
\Downarrow \\
\forall 1 \leq j \leq n : \mathtt{MI}_1(E_j) \rightarrow^* \mathtt{OK} \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \\
\Downarrow \\
(\mathtt{i}, \mathtt{m}) : (\mathtt{i}, (\mathtt{q}, \mathtt{a}), \mathtt{m}) \in \mathtt{Marked} = \mathtt{Assigned}
\end{gathered}$$

We can make a similar argument for completeness:

$$\begin{gathered}
(\mathtt{i}, \mathtt{m}) : (\mathtt{i}, (\mathtt{q}, \mathtt{a}), \mathtt{m}) \in \mathtt{Marked} = \mathtt{Assigned} \\
\Downarrow \\
\forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \\
\Downarrow_{(using\ ProVerif)} \\
\forall 1 \leq j \leq n : \mathtt{MI}_1(E_j) \rightarrow^* \mathtt{OK} \\
\Downarrow \\
\mathtt{MI}_n(E) \rightarrow^* \mathtt{OK}
\end{gathered}$$

3.4.3.2 Protocol by Giustolisi *et al.*

The second exam protocol is Remark! [GLR14], we described in Section 3.3.3.2.

Formal Model We use the same equational theory presented in Figure 3.5. Data sets I, Q, A and M are as in Definition 3.13. Set I contains the C's pseudonyms rather than their identities. This replacement is sound because any candidate is uniquely identified by her pseudonym and the equational theory preserves this bijection. The sets Q, A, and M are the messages that correspond to questions, answers, and marks generated during the protocol's run. The relations are built from the posts that appear on the BB. Precisely, the tuple $(i, (q, a))$ of `Accepted` is built from the receipts that EA publishes at Examination. The tuples $(i, (q, a), m)$ and (i, m) of `Marked` and `Assigned` consist of the posts that EA publishes at Marking. Precisely, the tuple $(i, (q, a), m)$ is built from the marked exam-form signed by E, while the tuple (i, m) is built from the encryption of the marked exam-form that EA generates. In fact, the encryption requires a pseudonym, and officially links C with their identities. This replacement is sound because C is uniquely identified by her key and the marked exam-form. `Correct` is the algorithm used to mark the exam-forms and is modeled using a table.

Analysis of Individual Verifiability. Similarly to what we did in the analysis of the Grenoble case, we modeled our individual verifiability tests in ProVerif. We used assertions to prove soundness, and unreachability of an event `KO` to prove completeness. In checking the soundness of a test we assumed, in addition to the honest NET, a honest C (the verifier). The roles of E and co-candidate are dishonest for all tests. The input of a test consists of the data sent via private channel from C, the data sent via public channel from EA, and the evidences posted on BB. To check the completeness of a test, we model all roles as honest. They all send their data via private channel to the test, whose input also includes the evidences posted on BB.

Remark! protocol originally mandates only two individual verifiability properties: Mark Notification Integrity and a weaker version of Exam-form Integrity. However, we checked which assumptions Remark! protocol needs in order to ensure all our properties. For each property, we describe how to build the corresponding test. Table 3.6 summarizes the ProVerif results of our analysis.

Question Validity: the test (Figure 3.20) checks if the question given to the candidate is correctly signed by the exam authority. Since the questions are generated by the Exam Authority, it is modelled as an honest role. To check soundness in ProVerif, we check if the test emits the event `OK` only if the Exam Authority actually generated the question, *i.e.*, any event `OK` is preceded by the event `generated`. ProVerif shows that Remark! is question validity verifiable.

Marking Correctness: Remark! does not originally provide the marking algorithm used to evaluate the answer. Consequently, there exists no test that candidate can run to verify Marking Correctness. However, we prove in ProVerif that if the (honest) exam authority publishes the table of evaluations after the exam concludes, the property holds

```
let testQV(pkA, pch, bba)=
in(bba, eques);
in(pch, (ques, priv_C));
let (ques', sques) = decrypt(eques, priv_C) in
let (ques'', pseudoC) =checksign(sques, pkA) in
if ques'=ques && ques''=ques'
then
event OK
else
event KO.
```

FIGURE 3.20: The Question Validity test for Remark! protocol.

(both soundness and completeness). Given the exam-form submitted by the candidate and the table of evaluations, the test (Figure 3.21) checks if the mark reported on table that corresponds to the exam-form coincides with the mark notified to the candidate.

```
let testMC (pkA, pch) =
in(pch, (ques: bitstring,ans: bitstring, mark: bitstring));
get correct_ans(=ques,=ans,mark') in
if mark'=mark
then
event OK
else
event KO.
```

FIGURE 3.21: The Marking Correctness for Remark!.

Correct Mark Reception: the Correct Mark Reception test (Figure 3.24) checks if the exam-form submitted and the mark received by the candidate (`ques`,`ans`), and the notification (`ema'`) published on the bulletin board by the exam authority, contain the same questions, answers, pseudonym, and mark. For soundness, we label the corresponding ProVerif test code with two events (`accepted` and `marked`), which map the corresponding relations. In particular, we consider a receipt part of the relation `Accepted` if it is signed by the exam authority and encrypted under the pseudonym of the candidate. Similarly, we consider a notification part of the relation `Marked` if it is signed by the examiner and encrypted under the pseudonym of the candidate. ProVerif shows that the test for Correct Mark Reception is sound and complete.

Exam-form Integrity: for Remark! protocol the Exam-form Integrity test (Figure 3.24) checks if the exam-form submitted by the candidate (`ques`,`ans`), the corresponding receipt (`eca'`), and the notification (`ema'`) published on the bulletin board by the exam authority, contain the same questions, answers, and pseudonym. For soundness, we label the corresponding ProVerif test code with the event (`accepted`), which map the corresponding relation. In particular, we consider a receipt part of the relation `Accepted`

```
let testCMR (pkN, pkA, pch, bbn, bba)=
in(pch, (pseudo_C, ques,ans, mark, priv_C));
in(bbn, (pseudo_E, he, rolet: role, spseE));
in(bba, ema');
let (((ques', ans', pseudo_C'),sca1, mark'), sma)
                                        = decrypt(ema', priv_C) in
let ((ques'', ans'', pseudo_C''),sca1', mark'')
                                        = checksign(sma, pseudo_E) in
if ques'=ques && ans'=ans && pseudo_C'=pseudo_C && mark'=mark &&
    (ques', ans', pseudo_C')=checksign(sca1,pkA) && ques''=ques &&
    ans''=ans && pseudo_C''=pseudo_C && mark''=mark' && sca1'=sca1
then
event OK
else
event KO.
```

FIGURE 3.22: The Correct Mark Reception test for Remark! protocol.

if it is signed by the exam authority and encrypted under the pseudonym of the candidate. Note that a corrupted exam authority can publish two different receipts for the same exam-form 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. ProVerif shows that the test for Exam-form Integrity is sound and complete.

```
let testETI (pkN, pkA, pch, bbn, bba)=
in(pch, (pseudo_C, ques,ans, priv_C));
in(bbn, (pseudo_E, he, rolet: role, spseE));
in(bba, eca');
in(bba, ema');
let (ca, sca')=decrypt(eca', priv_C) in
let (ques', ans', pseudo_C')= checksign(sca', pkA) in
if  ques'=ques && ans'=ans && pseudo_C'=pseudo_C
then
event OK
else
event KO.
```

FIGURE 3.23: The Exam-form Integrity test for Remark! protocol.

Exam-form Markedness: the Exam-form Markedness test (Figure 3.24) checks if the exam-form submitted by the candidate (`ques`,`ans`), the corresponding receipt (`eca'`), and the notification (`ema'`) published on the bulletin board by the exam authority, contain the same questions, answers, and pseudonym. For soundness, we label the corresponding ProVerif test code with two events (`accepted` and `marked`), which map the corresponding

relations. In particular, we consider a receipt part of the relation `Accepted` if it is signed by the exam authority and encrypted under the pseudonym of the candidate. Similarly, we consider a notification part of the relation `Marked` it is signed by the examiner and encrypted under the pseudonym of the candidate. ProVerif shows that the test for Exam-form Markedness is sound and complete.

```
let testETI (pkN, pkA, pch, bbn, bba)=
in(pch, (pseudo_C, ques,ans, priv_C));
in(bbn, (pseudo_E, he, rolet: role, spseE));
in(bba, eca');
in(bba, ema');
let (ca, sca')=decrypt(eca', priv_C) in
let (ques', ans', pseudo_C')=checksign(sca', pkA) in
let (((ques'', ans'', pseudo_C''),sca1, mark), sma)
                                    =decrypt(ema', priv_C) in
let ((ques''', ans''', pseudo_C'''),sca1', mark')
                                    =checksign(sma, pseudo_E) in
if  (ques', ans', pseudo_C')=checksign(sca1,pkA) &&
    pseudo_C'=pseudo_C && ques''=ques' && ans''=ans' &&
    pseudo_C''=pseudo_C && ques'''=ques' && ans'''=ans' &&
    pseudo_C'''=pseudo_C' && sca1'=sca1
then
event OK
else
event KO.
```

FIGURE 3.24: The Exam-form Markedness test for Remark! protocol.

Mark Integrity: similarly to Exam-form Integrity, we can show that Remark! is mark integrity verifiable. Given the exam-form submitted by the candidate (`ques`,`ans`) and the corresponding notification (`ema'`) published by the exam authority, the test (Figure 3.25) checks if they contain the same questions, answers, and pseudonym and that the examiner's signature on the mark is correct. To check soundness in ProVerif, we label the corresponding test code with two events (`assigned` and `marked`), which map the corresponding relations with respect to the notification published on the bulletin board. We consider the notification part of `Assigned` if it is signed by the exam authority and encrypted under the pseudonym of the candidate. Similarly, we consider the notification part of the relation `Marked` if it also includes the signature of the examiner.

Mark Notification Integrity: given the mark notified to the candidate and the corresponding notification (`ema'`) published by the exam authority, the test (Figure 3.26) simply checks if the marks coincide. Similarly to Mark Integrity, we consider the notification part of `Assigned` if it is signed by the exam authority and encrypted under the pseudonym of the candidate. ProVerif shows that the test for Exam-form Integrity is sound and complete.

```
let testMI (pkN: pkey, pkA: pkey, pch, bbn, bba)=
in(pch, (pseudo_C, ques, ans, priv_C));
in(bbn, (pseudo_E, he, rolet:role, spseE));
in(bba, ema');
let (((ques', ans', pseudo_C'),sca', mark), sma)
                                        = decrypt(ema', priv_C) in
let ((ques'', ans'', pseudo_C'': pkey),sca'', mark')
                                        =checksign(sma, pseudo_E) in
if  pseudo_C'=pseudo_C && pseudo_C''=pseudo_C && mark=mark' &&
    (ques', ans', pseudo_C')=checksign(sca',pkA)
then
event OK
else
event KO.
```

FIGURE 3.25: The Mark Integrity test for Remark! protocol.

```
let testMNI (pkA, priv_ch, bba)=
in(priv_ch, (priv_C, mark, pseudo_C));
in(bba, ema');
let (((ques, ans, pseudo_C'), sca, mark'), sma)
                                         =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 3.26: The Mark Notification Integrity test for Remark! protocol.

Analysis of Universal verifiability. We check most of the universal verifiability tests using a different approach compared to the individual ones. This is needed because C can be dishonest, in contrast to the case of individual verifiability, thus no sufficient events can be insert in any process to model correspondence assertions. In general, the idea of this approach is that every time the 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 `bad` is emitted, and we check soundness of the tests using unreachability of the event `bad`. However, we can still model soundness using correspondence assertions for Registration, because the NET is honest and emits an event when registration concludes. Since all the bulletin board posts are encrypted with C's or E's pseudonyms, no public data can be used as it is. Moreover, the encryption algorithm has the probabilistic feature of ElGamal encryption, thus the random value used to encrypt a message is usually

Property	Sound	Complete
Question Validity	✓ (EA)	✓(all)
Marking Correctness	✓ (EA)	✓(all)
Correct Mark Reception	✓	✓(all)
Exam-form Integrity	✓	✓(all)
Exam-form Markedness	✓	✓(all)
Mark Integrity	✓	✓(all)
Mark Notification Integrity	✓	✓(all)

TABLE 3.6: Results for I.V. properties of the Remark! protocol.

needed. However, an auditor can access some data posted by EA after the exam concludes. Precisely, we assume that the auditor is given the following data:

1. For Registration, the EA reveals the signatures inside the receipts posted on BB and the random values used to encrypt the receipts. By looking at the bulletin board, the auditor can check that EA only accepted tests signed with pseudonyms posted by the NET during registration.

2. For Exam-form Integrity, the EA reveals the marked exam-form and the random values used to encrypt them, in addition to the data given for Registration. In so doing, the auditor can check if the pseudonyms, questions, and answers did not change and got marked.

3. For Exam-form Markedness, the auditor accesses the same data outlined above for Exam-form Integrity. However, since Remark! is exam-form integrity universally verifiable, it is easy to show that the protocol is exam-form markedness universally verifiable, too.

4. For Marking Correctness, the EA reveals the marked exam-form, the random values used to encrypt the marked exam-form, and a table that maps a mark to each answer, after the exam concludes.

According our test codes outlined in Figures 3.27-3.28, the data needed to make Remark! testable is received from channel `ch`. As already mentioned for the Grenoble exam, ProVerif is unable to handle the general case for the universal verifiability properties. We thus prove in ProVerif the case with one candidate and rely on the general manual proofs seen in section 3.4.3.1 for the case with more candidates. Table 3.7 summarizes the results of our analysis.

Registration: Remark! ensures Registration if the exam authority reveals to the third party the signatures inside the receipts (posted on the bulletin board) and the random values used to encrypt the receipts after the exam concludes. In doing so, the third party, who run the test in Figure 3.27, can check by looking at the bulletin board that the

exam authority only accepted tests signed with pseudonyms posted by the NET during registration.

Exam-form Integrity: we prove that Remark! is exam-form integrity universally verifiable if the exam authority reveals the signatures inside the receipts and on the notification posted on the bulletin board, and the random values used for the encryption under the candidate pseudonyms. Then, the third party, can run the test in Figure 3.28 to check if the pseudonyms, questions, and answers did not change and get marked. For simplicity, we assume that only one examiner marks the exam-forms.

Exam-form Markedness: since Remark! is exam-form integrity universally verifiable, it is easy to show that it is also exam-form markedness universally verifiable.

Marking Correctness: Remark! does not originally provide the marking algorithm used to evaluate the answer. However, ProVerif shows that Remark! can be marking correctness universally verifiable by a third party by running the test in Figure 3.29, provided that the honest exam authority reveals 1) the tests and the marks encrypted on the notifications (posted on the bulletin board), 2) the random values used for the encryption under the candidate pseudonyms, and 3) the marking algorithm after the exam concludes. Similarly to Exam-form Integrity, we assume that one examiner marks all the exam-forms for simplicity.

Mark Integrity: regarding this property, we note that the input needed for this test (Figure 3.30) is the notification posted on the bulletin board by the exam authority. In fact, the examiner should reveal the signature of the examiners on the notification posted on the bulletin board, and the random values used to encrypt under the candidate pseudonyms. In so doing the third party can verify that each mark notified to the candidate has a correct signature.

Property	Sound	Complete
Registration	✓	✓(all)
Exam-form Integrity	✓	✓(all)
Exam-form Markedness	✓	✓(all)
Marking Correctness	✓ (EA)	✓(all)
Mark Integrity	✓	✓(all)

TABLE 3.7: Results for U.V. properties of the Remark! protocol.

```
let testR(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 (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 KO.
```

FIGURE 3.27: The universal Registration test for Remark! protocol.

3.5 Monitoring Exams

In the previous section, we provided an abstract definition of exam verifiability. Although it is useful to analyze exams as we have shown with our case studies, such approach is limited to exam specifications and does not allow us to analyze real exam implementations. Thus, it cannot deal with some flaws such that those results from the errors that may have been introduced at implementation stage or during execution. Moreover, scalability limitations may be faced when considering complex large systems and large number of parties. In this section, propose several monitors that allow us to verify real exam execution at runtime. Although, results of runtime monitoring are limited to the analyzed

```
let testETI(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 3.28: The universal Exam-form Integrity test for Remark! protocol.

exam runs, it has an advantage that the actual behavior is analyzed. We first formalize several exam requirements as Quantified Event Automata (QEAs). Then, we perform an offline monitoring, using MarQ tool [RCR15], of real e-exam executions organized by *Université Joseph Fourier* (UJF). Namely, for each exam execution, we extract a sequence of events (a trace). Then we feed the extracted trace to a monitor constructed using MarQ tool, which processes the trace and produces a verdict based on the defined requirements.

```
let testMC (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 3.29: The universal Marking Correctness test for Remark! protocol.

```
let testMI (bbn, bba1,...,bban, ch1,...,chn) =
in(bbn, (pseudo_E, he, re, spseE));
in(bba1, notif1);
...
in(bban, notifn);
in(ch1, (rcoin1, sma1));
...
in(chn, (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 3.30: The universal Mark Integrity test for Remark! protocol.

3.5.1 Exam Run and Events

We define an *e-exam run* (or *e-exam execution*) by a finite sequence of events, called *trace*. Such event-based modelling of e-exam runs is appropriate for monitoring actual events of the system. An exam run satisfies a property if the resulting trace is accepted by the corresponding monitor. A *correct exam run* satisfies all the properties. In order to formalize exam requirements, we consider the events $\texttt{register}(i)$: i registered to the exam; $\texttt{submit}(i, q, a)$: i submitted (q, a); and $\texttt{accept}(i, q, a)$: (q, a) accepted by the authority from i, that are already defined in Section 3.3.1. However, from now on we assume that the question q and the answer a respectively refer only to one question and one answer. We also consider the following events:

- Event $\texttt{get}(i, q)$ is emitted when candidate i gets question q.
- Event $\texttt{change}(i, q, a)$ is emitted when candidate i changes the answer field of question q to a.
- Event $\texttt{corrAns}(q, a)$ is emitted when the authority specifies a as a correct answer to question q. Note that more than one answer can be correct for a given question.
- Event $\texttt{marked}(i, q, a, b)$ is emitted when the answer a from candidate i to question q is scored with b. We assume that the score b ranges over $\{0, 1\}$ (1 for correct answer and 0 for wrong answer). However, other ranges of scores can be considered. Note that, event $\texttt{marked}(i, q, a, b)$ is similar to event $\texttt{attribut}(ques, ans, m, id_{form}, ide)$ already defined in Section 3.3.1. However, event `attribut` means that an (overall) mark is attributed to an exam-form instead of one question/answer pair in case of `marked` event. Moreover, `marked` specifies the identity of the candidate i while `attribut` specifies the identity of the examiner.
- Event $\texttt{assign}(i, m)$ is emitted when the mark m is assigned to the candidate i. Note that, event $\texttt{assign}(i, m)$ is different from event $\texttt{notify}(i, m)$ defined in Section 3.3.1. Instead it corresponds to the set `Assigned` defined in Section 3.4 as event `assign` signifies that an authority associates a mark m with candidate i, while the `notify` signifies that the candidate i received the mark m.
- Event $\texttt{begin}(i)$ is emitted when candidate i begins the examination phase.
- Event $\texttt{end}(i)$ is emitted when candidate i ends the examination phase. The candidate terminates the exam himself, *e.g.*, after answering all questions before the end of the exam duration.

In general, all events are time stamped, however we parameterize them with time only when it is relevant for the considered property. Moreover, we may omit some parameters

from the events when they are not relevant to the property. For instance, we may use `submit`(i) when candidate i submits an answer regardless of his answer. Note that, the events here are assumed to be recorded during the exam or built from data logs.

3.5.2 E-exams Requirements

In this section, we define the following eight properties that aim at ensuring e-exams correctness:

- *Candidate Registration*: no unregistered candidate tries to participate in the exam by submitting an answer.
- *Candidate Eligibility*: answers are accepted only from registered candidates.
- *Answer Authentication*: all accepted answers are actually submitted by candidates, and for each question at most one answer is accepted *per* candidate.
- *Questions Ordering*: all candidates answer the questions in the required order.
- *Exam Availability*: answers are accepted only during the examination time.
- *Exam Availability with Flexibility*: a variant of *Exam Availability* that supports both flexible starting time and duration of the exam.
- *Marking Correctness*: all answers are marked correctly.
- *Mark Integrity*: each candidate assigned his mark.

Each property is defined as a QEA according to the formalism introduced in Section 2.3. Note that, we extend the initial definition of QEAs in [BFH+12] by i) allowing variable declaration and initialization before reading the trace, and ii) introducing the notion of global variable shared among all event automaton instances. Global variables are mainly needed in QEAs to keep track and report data at the end of monitoring.

Note that, each property represents a different e-exam requirement and can be monitored independently. An *exam run* may satisfy one property and fail on another one, which narrows the possible source of potential failures and allows to deliver a detailed report about the satisfied and unsatisfied properties. Note also that we assume that an input trace contains events related to a single exam run. To reason about traces with events from more than one exam run, the events have to be parameterized with an exam run identifier, which has to be added to the set of quantified variables.

3.5.2.1 Properties for Error Detection:

Candidate Registration: states that only already registered candidates can submit answers to the exam. An exam run satisfies *Candidate Registration* if, for every candidate i, event `submit`(i) is preceded by event `register`(i). A violation of *Candidate Registration* does not reveal a weakness in the exam system (as long as the answers submitted from unregistered candidates are not accepted by the authority). However, it allows us to detect if a candidate tries to fake the system, which is helpful to be aware of *spoofing* attacks.

Definition 3.27. (Candidate Registration). Candidate Registration *is defined by the QEA depicted in Figure 3.31 with alphabet* $\Sigma_{\text{CR}} = \{\texttt{register}(i), \texttt{submit}(i)\}$.

FIGURE 3.31: A QEA for Candidate Registration.

The input alphabet Σ_{CR} for *Candidate Registration* contains only events `register`(i) and `submit`(i), so any other events in the trace are ignored for this property. The QEA for *Candidate Registration* has two accepting states, and one quantified variable i. As the initial state is an accepting state, then the empty trace is accepted by the QEA. State (1) is a square state, so an event `submit`(i) that is not preceded by event `register`(i) leads to a failure. An event `register`(i) in state (1) leads to state (2) which is a skipping (circular) state. Henceforth, given a candidate i any trace starting with event `register`(i) is accepted.

The quantification $\forall i$ means that the property must hold for all values that i takes in the trace, *i.e.*, the values obtained when matching the symbolic events in the specification with concrete events in the trace. For instance, consider the following trace: `register`(i_1).`submit`(i_2).`submit`(i_1) .`register`(i_2). To decide whether it is accepted or not, the trace is sliced based on the values that can match i, resulting in two slices: $i \mapsto i_1$: `register`(i_1).`submit`(i_1), and $i \mapsto i_2$: `submit`(i_2).`register`(i_2). Then, each slice is checked against the event automaton instantiated with the appropriate value for i. The slice associated to i_1 is accepted as it reaches the final state (2), while the slice associated to i_2 does not reach a final state since event `submit`(i_2) leads from state (1) to an implicit failure state. Therefore, the whole trace is not accepted by the QEA. Note that, we omit parameters q and a from event `submit`(i, q, a) since only the fact that a candidate i submits an answer is significant for the property, regardless of the question he is answering, and the answer he submitted.

Candidate Eligibility: states that no answer is accepted from an unregistered candidate.

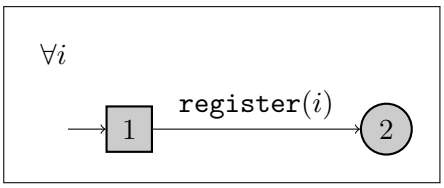


FIGURE 3.32: A QEA for Candidate Eligibility.

It is modeled by a QEA similar to that of *Candidate Registration* depicted in Figure 3.31, except that event $\texttt{submit}(i, q, a)$ has to be replaced by $\texttt{accept}(i, q, a)$ in the related alphabet.

Definition 3.28. (Candidate Eligibility). Candidate Eligibility *is defined by the QEA depicted in Figure 3.32 with alphabet* $\Sigma_{\text{CE}} = \{\texttt{register}(i), \texttt{accept}(i, q, a)\}$.

Trace $\texttt{register}(i_1).\texttt{accept}(i_2, q_0, a_2).\texttt{accept}(i_1, q_0, a_1).\texttt{register}(i_2)$ is not accepted by *Candidate Eligibility* as an answer is accepted from the candidate $\texttt{i}_2$ before he registered.

Answer Authentication: states that all accepted answers are submitted by candidates. Moreover, for every question, exactly one answer is accepted from each candidate that submitted at least one answer to that question.

Definition 3.29. (Answer Authentication). Answer Authentication *is defined by the QEA depicted in Figure 3.33 with alphabet* $\Sigma_{\text{AA}} = \{\texttt{submit}(i, q, a), \texttt{accept}(i, q, a)\}$.

The QEA of *Answer Authentication* fails if an unsubmitted answer is accepted. A candidate can submit more than one answer to the same question, however exactly one answer has to be accepted. Note that, any answer among the submitted answers can be accepted. However, the QEA can be updated to allow only the acceptance of the last submitted answer by replacing set $\texttt{A}$ with a variable, which acts as a placeholder for the last submitted answer. If no answer is accepted after at least one answer has been submitted, the QEA ends in the failure state (2), while acceptance of an answer leads to the accepting state (3). A candidate can submit after having accepted an answer from him to that question. However, if more than one answer is accepted, an implicit transition from state (3) to a failure state is fired. The QEA of *Answer Authentication* accepts the trace $\texttt{submit}(i_1, q_0, a_1).\texttt{submit}(i_1, q_0, a_2).\texttt{accept}(i_1, q_0, a_2)$, where candidate i_1 submits two answers a_1 and a_2 to question q_0, then only a_2 is accepted. While it rejects the traces $\texttt{accept}(i_1, q, a)$, where an unsubmitted answer is accepted from i_1, and $\texttt{submit}(i_1, q, a_1)$. $\texttt{submit}(i_1, q, a_2)$. $\texttt{accept}(i_1, q, a_1).\texttt{accept}(i_1, q, a_2)$, where two answers to the same question are accepted from same candidate. *Answer Authentication* can be further split into three different properties which allow us to precisely know whether, only submitted answers are accepted (*Answer Submission Authentication*), for every question an answer is accepted from a candidate that submitted at least one answer (*Acceptance*

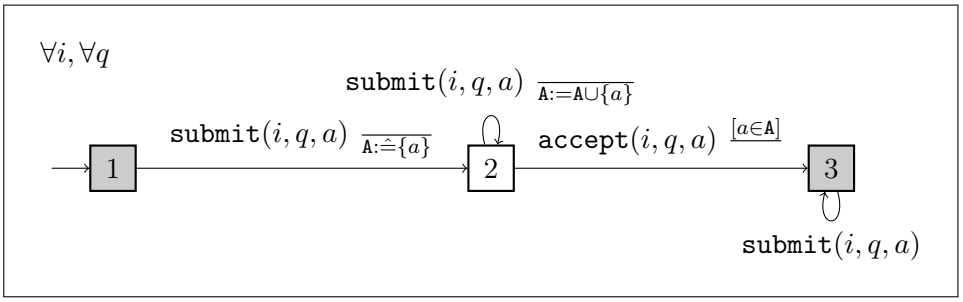


FIGURE 3.33: A QEA for Answer Authentication.

Ensurance), and only one answer is accepted from the same candidate for the same question (*Answer Singularity*).

Definition 3.30. (Answer Submission Authentication). Answer Submission Authentication *is defined by the QEA depicted in Figure 3.33 with alphabet* $\Sigma_{\text{ASA}} = \{\texttt{submit}(i,q,a), \texttt{accept}(i,q,a)\}$.

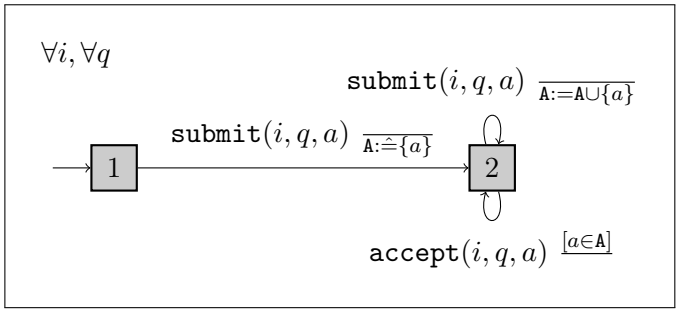


FIGURE 3.34: A QEA for Answer Submission Authentication.

The QEA presented in Figure 3.34 fails when an unsubmitted answer is accepted from a certain candidate regardless of whether that candidate submitted a different answer or not. Traces $\texttt{accept}(i_1,q_1,a_1)$ and $\texttt{submit}(i_1,q_1,a_1).\texttt{accept}(i_1,q_1,a_2)$ are both unaccepted traces by QEA of Figure 3.34. In the former an answer for the question q_1 is accepted from candidate i_1, which did not submit any answer to question q_1. In the latter the candidate i_1 submitted an answer a_1 while a different answer a_2 is accepted from him. Note that, it is allowed that no answer is accepted from a candidate, or multiple answers are accepted from the same candidate as long as they are all submitted by that candidate.

Definition 3.31. (Acceptance Ensurance). Acceptance Ensurance *is defined by the QEA depicted in Figure 3.35 with alphabet* $\Sigma_{\text{AE}} = \{\texttt{submit}(i,q,a), \texttt{accept}(i,q,a)\}$.

The QEA presented in Figure 3.35 fails when no answer is accepted from a candidate that submitted at least one answer. Note that, the QEA succeeds even if the accepted answer is not one of the submitted answers. The latter case leads to failure of QEA presented in Figure 3.34.

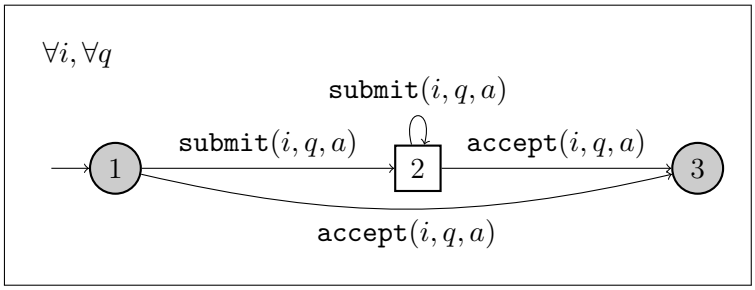


FIGURE 3.35: A QEA for Acceptance Ensurance.

Definition 3.32. (Answer Singularity). Answer Singularity *is defined by the QEA depicted in Figure 3.35 with alphabet* $\Sigma_{\text{AS}} = \{\texttt{submit}(i, q, a), \texttt{accept}(i, q, a)\}$.

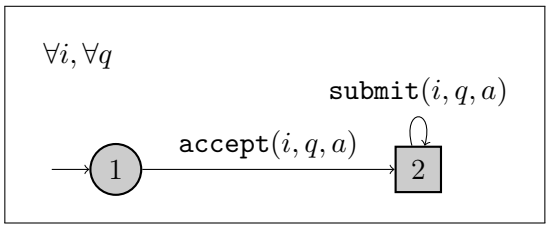


FIGURE 3.36: A QEA for Answer Singularity.

The QEA presented in Figure 3.36 fails if more than one answer is accepted from the same candidate to the same question, even if all the accepted answers are submitted ones. Traces $\texttt{accept}(i_1, q_1, a_1)$ and $\texttt{submit}(i_1, q_1, a_1).\texttt{accept}(i_1, q_1, a_1)$ are accepted traces. While trace $\texttt{submit}(i_1, q_1, a_1).\texttt{accept}\ (i_1, q_1, a_1).\texttt{submit}(i_1, q_1, a_2).\texttt{accept}(i_1, q_1, a_2)$ is an unaccepted trace as two answers are accepted from the same candidate to the same question.

Questions Ordering: states that a candidate should not get a question before validating his answer to the previous question. Note that, the previous properties formalize the main requirements that are usually needed concerning answer submission and acceptance. However, *Questions Ordering* might be required as an additional requirement.

Definition 3.33. (Question Ordering). *Let* $q_1, \ldots, q_n$ *be* n *questions such that the order* $\texttt{ord}(q_k)$ *of* q_k *is* k. Questions Ordering *is defined by the QEA depicted in Figure 3.37 with alphabet* $\Sigma_{\text{QO}} = \{\texttt{get}(i, q), \texttt{accept}(i, q)\}$.

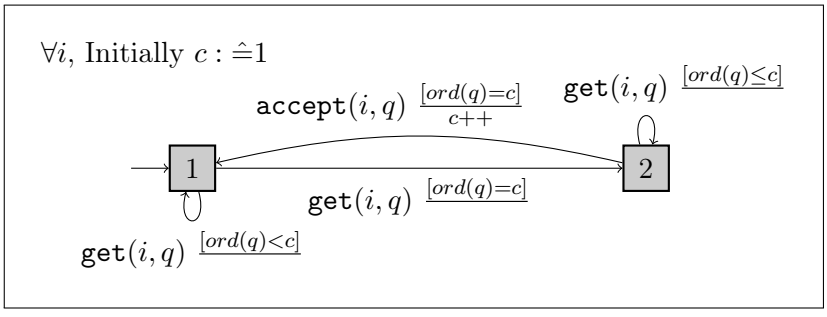


FIGURE 3.37: A QEA for Questions Ordering.

The QEA of *Questions Ordering* fails if a candidate gets or an answer is accepted from him for a higher order question before his answer to the current question is accepted. This is ensured by the guard $[\texttt{ord}(q) = c]$ on the self loop transition on state (2). Note that, *Questions Ordering* also allows only one accepted answer per question. Otherwise, there is no meaning for the order as the candidate can re-submit answers latter when he gets all the questions.

Exam Availability: states that questions are obtained, and answers are submitted and accepted only during the examination time. *Exam Availability* is necessary to ensure that all candidates have took the exam during the examination phase.

Definition 3.34. (Exam Availability). *Let t_0 and t_f be the starting and ending time instants of the exam, respectively.* Exam Availability *is defined by the QEA depicted in Figure 3.38 with alphabet* $\Sigma_{\text{EA}} = \{\texttt{get}(i,t), \texttt{change}(i,t), \texttt{submit}(i,t), \texttt{accept}(i,t)\}$.

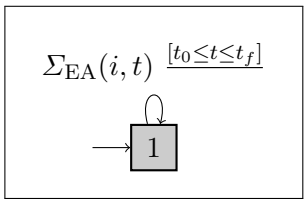


FIGURE 3.38: A QEA for Exam Availability.

The QEA of *Exam Availability* checks that all the events in Σ_{EA} are emitted between t_0 and t_f. Note that, any other event can be added to Σ_{EA} if required.

Exam Availability with Flexibility: is a variant of *Exam Availability* that supports exams with flexible starting and duration time. Some exams offer flexibility to the candidates, so that a candidate is free to choose the beginning time within a certain specified period. To capture that, we define *Exam Availability with Flexibility* which states that no answer can be accepted from a candidate before he begins the exam, after he terminates the exam, after the end of his exam duration, or after the end of the specified period. The beginning time of the exam may differ from one candidate to another, but in any case it has to be within a certain specified period. The exam duration may also differ between candidates. For example, an extended duration may be offered to certain candidates with disabilities.

Definition 3.35. (Exam Availability With Flexibility). *Let t_1 and t_2 respectively be the starting and the ending time instants of the allowed period, and let* $\texttt{duration}(i)$ *be the exam duration for candidate i.* Exam Availability with Flexibility *is defined by the QEA depicted in Figure 3.39 with alphabet* $\Sigma_{\text{EAF}} = \{\texttt{begin}(i,t), \texttt{end}(i), \texttt{accept}(i,t)\}$.

Exam Availability with Flexibility also requires that, for each candidate i, there is only one event $\texttt{begin}(\texttt{i},\texttt{t})$ per exam. Hence, it fails if event $\texttt{begin}(i)$ is emitted more than once. A candidate can begin his exam at any time t_b such that $t_1 \leq t_b \leq t_2$. Note that, no answer

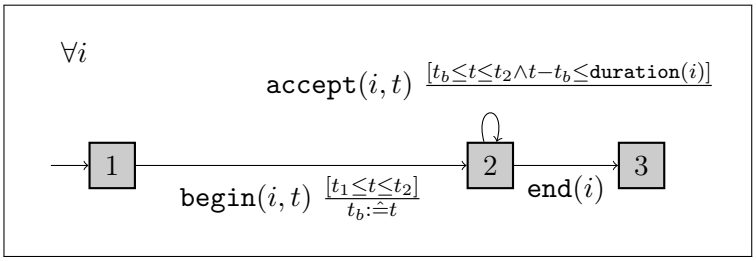


FIGURE 3.39: A QEA for Exam Availability with Flexibility.

can be accepted from a candidate after then ending time t_2 of the period, if the duration of the candidate is not finished yet. Assume that $t_1 = 0$, $t_2 = 1,000$, and that the exam duration of i_1 and i_2 respectively are $\texttt{duration}(i_1) = 90$ and $\texttt{duration}(i_2) = 60$. Then, trace $\texttt{begin}(i_1, 0).\texttt{accept}(i_1, 24).\texttt{begin}(i_2, 26).\texttt{accept}\ (i_2, 62).\texttt{accept}(i_1, 90)$ is accepted. While, trace $\texttt{accept}(i_1, 5).\texttt{begin}\ (i_1, 20)$ and trace $\texttt{begin}(i_1, 0).\texttt{accept}(i_1, 91)$ are not accepted since in the first one an answer is accepted from candidate i_1 before he begins the exam, and in the second one an answer is accepted after the exam duration expires. Event `submit` is not included in Σ_{EAF}, thus an answer submission outside the exam time is not considered as an irregularity if the answer is not accepted by the authority. However, again other events (*e.g.*, `get` and `submit`) can be considered. In such a case, the QEA in Figure 3.39 has to be edited by looping over state (2) with any added event.

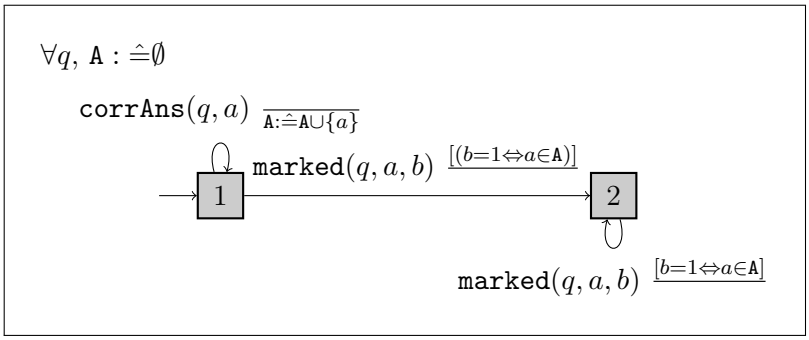


FIGURE 3.40: A QEA for Marking Correctness.

Marking Correctness: states that all answers are marked correctly. In the QEA of *Marking Correctness*, the correct answers for the considered question are collected in a set `A` (self loop over state (1)).

Definition 3.36. (Marking Correctness). *Property* Marking Correctness *is defined by the QEA depicted in Figure 3.40 with alphabet* $\Sigma_{\text{MC}} = \{\texttt{corrAns}(q, a), \texttt{marked}(q, a, b)\}$.

In state (1), once an answer to the considered question is marked correctly, a transition to state (3) is fired, otherwise if an answer is marked in a wrong way a transition to an implicit failure state occurs. In state (3), the property fails either if an answer is marked in a wrong way, or if an event $\texttt{corrAns}(q, a)$ is encountered as this means that certain answers are marked before all the correct answers are set.

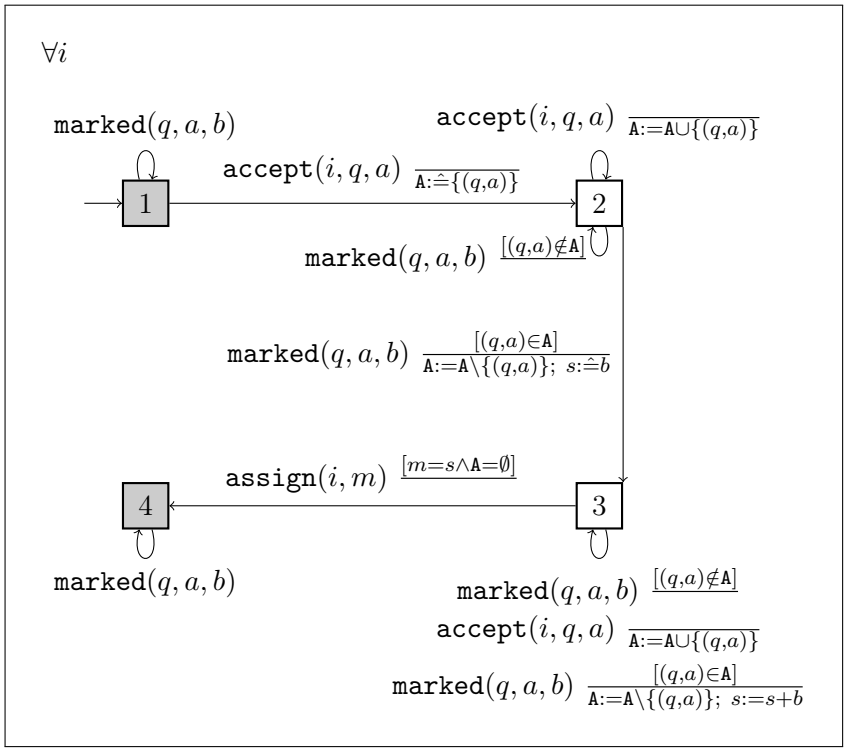


FIGURE 3.41: A QEA for Mark Integrity.

Mark Integrity: states that all the accepted answers are marked, and that exactly one mark is assigned to each candidate, the one attributed to his answers. *Mark Integrity* together with *Marking Correctness*, guarantees that each candidate participating in the exam gets the correct mark corresponding to his answers.

Definition 3.37. (Mark Integrity). *Property* Mark Integrity *is defined by the QEA depicted in Figure 3.41 with alphabet* $\Sigma_{\text{MI}} = \{\texttt{accept}(i, q, a), \texttt{marked}(\texttt{q}, \texttt{a}, \texttt{b}), \texttt{assign}(i, m)\}$.

The QEA of *Mark Integrity* collects, for each candidate, the answers that he submitted in set `A`. For each accepted answer, the QEA accumulates the corresponding score b in the sum s. If the accepted answers are not marked, the property fails (failure state (2)). If the candidate is not assigned a mark or assigned a wrong mark the property fails (failure state (3)). Once the correct mark is assigned to the candidate, if another mark is assigned or any other answer is accepted from him, the property fails (square state (4)).

3.5.2.2 Properties for Error Reporting

In the previous formalization, a property fails when its requirement is violated. However, it does not identify the entities that violate the requirement. In the following, we propose for each property an alternative that reports at the end all entities that violate the requirement of the property. In general, an alternative property ensures the same requirement(s) of the original regular property and has same input alphabet, but additionally reports some

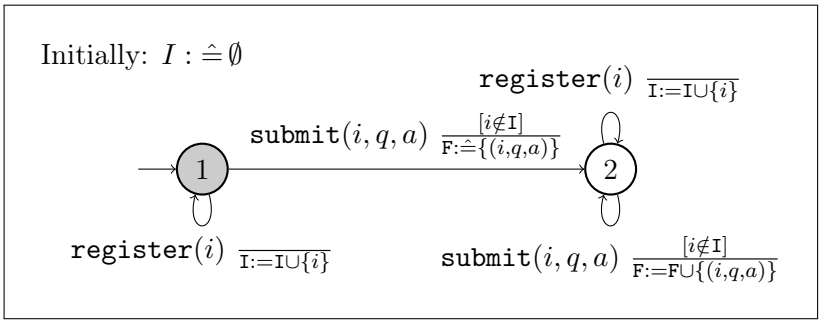


FIGURE 3.42: A QEA for Candidate Registration with Auditing.

data at the end. Whenever the requirements formalized by the following QEAs differ from their counterparts in the previous subsection, we mention it.

Candidate Registration with Auditing: fails if an unregistered candidate submitted an answer, and at the same time collects all such candidates in a set $\texttt{F}$.

Definition 3.38. (Candidate Registration with Auditing). Candidate Registration with Auditing *is defined by the QEA depicted in Figure 3.42 with alphabet* $\Sigma_{\text{CRA}} = \{\texttt{register}(i), \texttt{submit}(i)\}$.

The QEA of *Candidate Registration with Auditing* has three free variables $\texttt{I}, \texttt{F}$, and i, and no quantified variable. Instead of being instantiated for each candidate i, the QEA of *Candidate Registration with Auditing* collects all the registered candidates in set I, so that any occurrence of event $\texttt{submit}(i)$ at state (1) with $i \notin \texttt{I}$ fires a transition to the failure state (2). Such a transition results in the failure of the property since all transitions from state (2) are self-looping transitions. Set $\texttt{F}$ is used to collect all the unregistered candidates that submitted an answer, *i.e.*, those that violate the requirement. For example, trace $\texttt{register}(i_1).\texttt{submit}(i_2, q, a_2).\texttt{submit}(i_1, q, a_1)$ $.\texttt{register}(i_2)$ is not accepted by *Candidate Registration with Auditing*, and results in the set $\texttt{F} = \{(i_2, q, a_2)\}$.

Candidate Eligibility with Auditing: fails when an answer is accepted from an unregistered candidate, and reports all such candidates in a set $\texttt{F}$.

Definition 3.39. (Candidate Eligibility with Auditing). Candidate Eligibility with Auditing *is defined by the QEA depicted in Figure 3.43 with alphabet* $\Sigma_{\text{CEA}} = \{\texttt{register}(i), \texttt{accept}(i, q, a)\}$.

Note that, the QEA of *Candidate Eligibility with Auditing* is similar to the QEA of *Candidate Registration with Auditing*, except that the event $\texttt{submit}(i, q, a)$ is replaced by the event $\texttt{accept}(i, q, a)$.

Answer Authentication with Auditing: fails when an unsubmitted answer is accepted, no answer to a certain question is accepted from a candidate that submitted an answer for that question, or more than one answer is accepted from the same candidate to the same

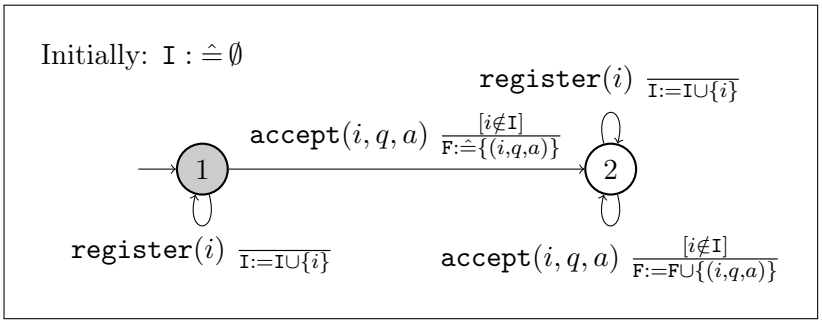


FIGURE 3.43: A QEA for Candidate Eligibility with Auditing.

question. The QEA of *Answer Authentication with Auditing* collects in a set $\mathtt{F_1}$ all the unsubmitted answers that are accepted together with the corresponding candidates and questions. It also collects in a set $\mathtt{F_2}$ the further accepted answers to the same question from the same candidate. Note that, when the first answer is accepted the free variable c is set to 1. The sets $\mathtt{F_1}$ and $\mathtt{F_2}$ are globally defined, and thus they are shared between all the instances of the QEA which result from the instantiation of the QEA for different i and q.

Definition 3.40. (Answer Authentication with Auditing). Answer Authentication *is defined by the QEA depicted in Figure 3.33 with alphabet* $\Sigma_{\text{AAA}} = \{\mathtt{submit}(i, q, a), \mathtt{accept}(i, q, a)\}$.

Questions Ordering with Auditing: states that all the candidates have to answer the questions in the required order. Additionally, it collects in a set $\mathtt{F}$ all candidates that get a higher order question before their answer to the current question is accepted. Also, a candidate which an answer is accepted from him for a question different from the current question is added to $\mathtt{F}$. Note that, the set $\mathtt{F}$ is a global set.

Definition 3.41. (Questions Ordering with Auditing). *Let* $q_1, \ldots, q_n$ *be* n *questions such that the order* $\mathtt{ord}(q_k)$ *of* q_k *is* k. Questions Ordering with Auditing *is defined by the QEA depicted in Figure 3.45 with alphabet* $\Sigma_{\text{QOA}} = \{\mathtt{get}(i, q), \mathtt{accept}(i, q)\}$.

Exam Availability with Auditing: checks that all the events in Σ_{EA} are emitted between t_0 and t_f. It also collects all the candidates that violate the requirements in a set $\mathtt{F}$.

Definition 3.42. (Exam Availability with Auditing). *Let* t_0 *and* t_f resp. *be the starting and finishing time of the exam.* Exam Availability with Auditing *is defined by the QEA depicted in Figure 3.46 with alphabet* $\Sigma_{\text{EAA}} = \{\mathtt{get}(i, t), \mathtt{submit}(i, t), \mathtt{accept}(i, t)\}$.

Exam Availability with Flexibility Auditing: collects all the candidates that violate the requirements of *Exam Availability with Flexibility* in a global set $\mathtt{F}$.

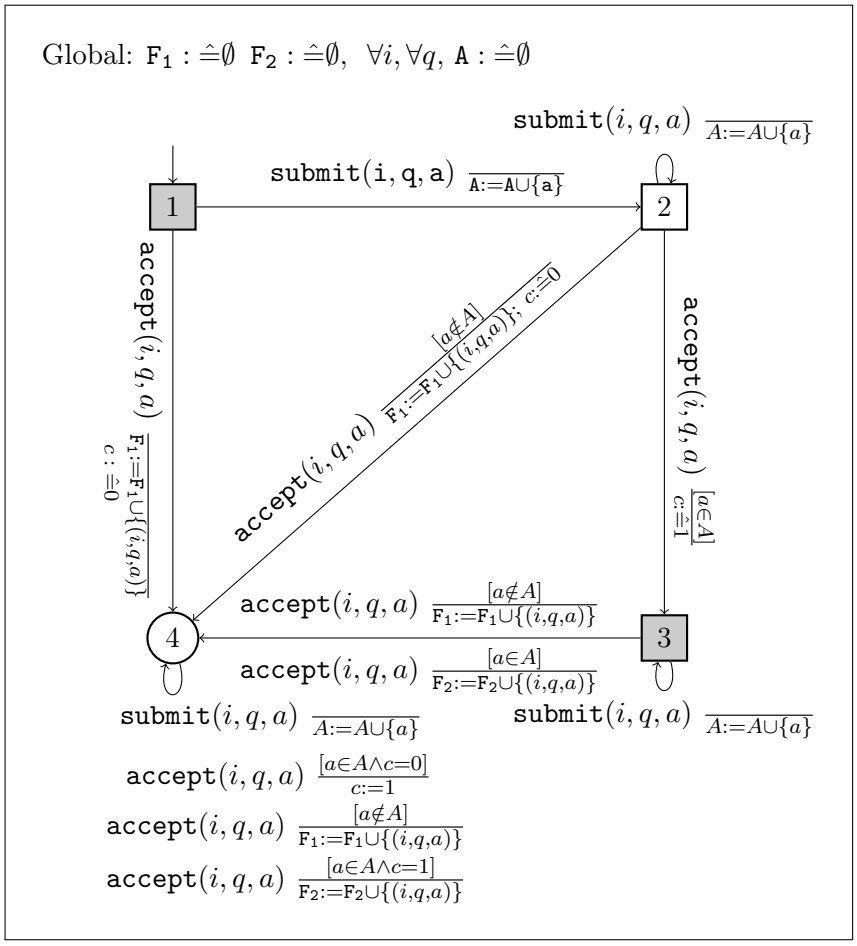


FIGURE 3.44: A QEA for Answer Authentication with Auditing.

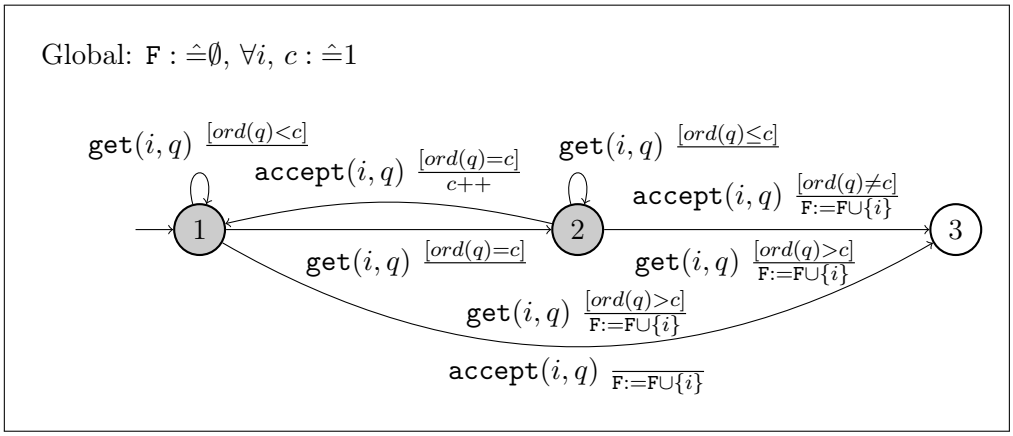


FIGURE 3.45: A QEA for Questions Ordering with Auditing.

Definition 3.43. (Exam Availability With Flexibility Auditing). *Let t_1 and t_2 respectively be the starting and the finishing of the allowed period, and let* $\mathtt{duration}(i)$ *be the exam duration of candidate i.* Exam Availability with Flexibility Auditing *is defined by the QEA depicted in Figure 3.47 with alphabet* $\Sigma_{\text{EAF}} = \{\mathtt{begin}(i,t), \mathtt{end}(i), \mathtt{accept}(i,t)\}$.

Marking Correctness with Auditing: collects all the answers that are marked in a wrong

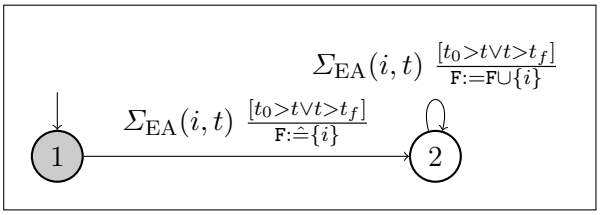


FIGURE 3.46: A QEA for Exam Availability with Auditing.

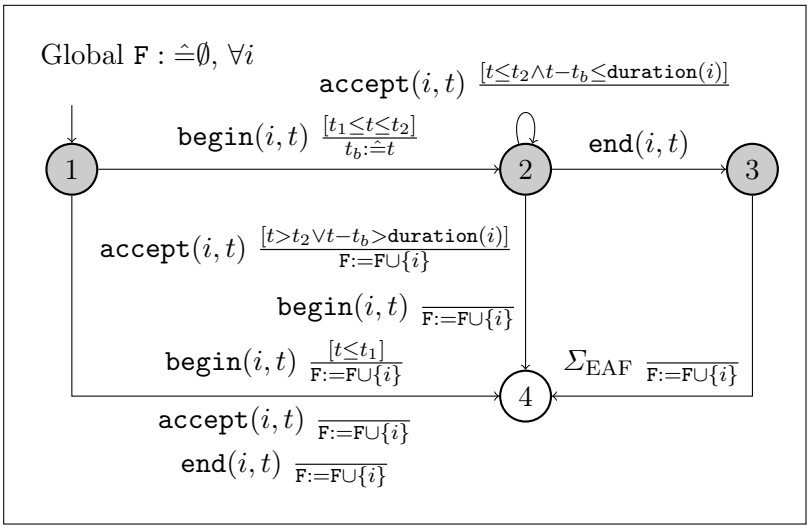


FIGURE 3.47: A QEA for Exam Availability With Flexibility Auditing.

way in a global set F. Note that, the fact that, for a question q, no event $\texttt{corrAns}(q,a)$ can be emitted after the marking of the first answer is marked is relaxed. Simply, an answer that is not declared as an correct answer yet is considered a wrong answer.

Definition 3.44. (Marking Correctness with Auditing). *Property* Marking Correctness with Auditing *is defined by the QEA depicted in Figure 3.48 with alphabet* $\Sigma_{\text{MC}} = \{\texttt{corrAns}(q,a), \texttt{marked}(q,a,b)\}$.

Mark Integrity with Auditing: states that all the accepted answers are marked, and that exactly one mark is assigned to each candidate, the one attributed to his answers. Note

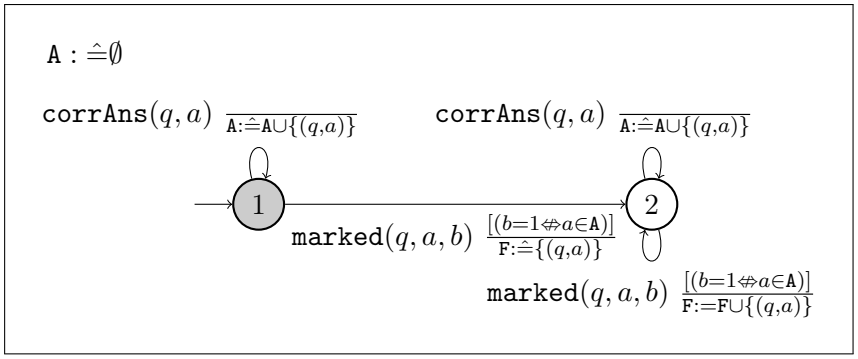


FIGURE 3.48: A QEA for Marking Correctness with Auditing.

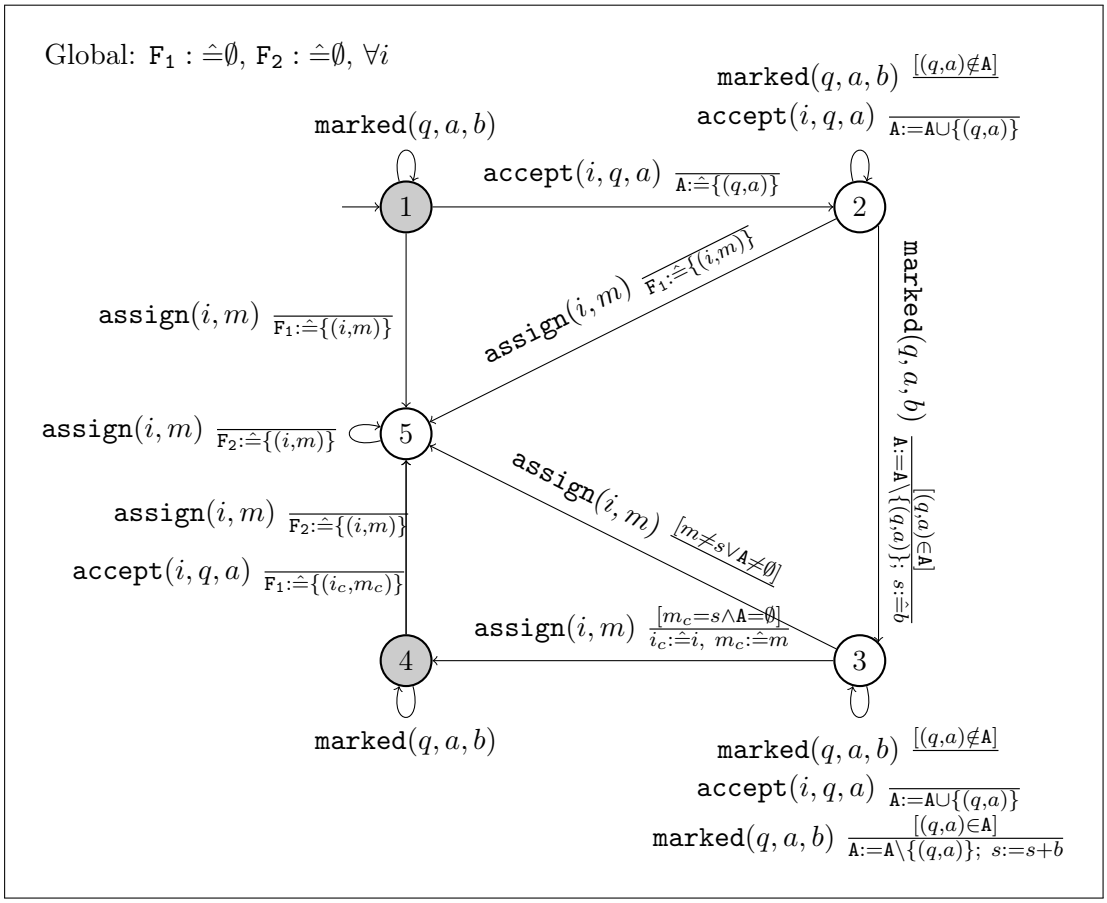


FIGURE 3.49: A QEA for Mark Integrity with Auditing.

that, a candidate that does not participated in the exam, *i.e.*, no answer is accepted from him, is assigned no mark. While, a candidate that participated in the exam but none of his answers is a correct answer is assigned the mark 0. Set $\mathtt{F_1}$ collects all the candidates that their first assigned mark is wrong. While, set $\mathtt{F_2}$ collects all the further marks assigned for the candidates regardless if they are correct or not.

Definition 3.45. (Mark Integrity with Auditing). Mark Integrity with Auditing *is defined by the QEA depicted in Figure 3.49 with alphabet* $\Sigma_{\text{MI}} = \{\mathtt{accept}(i,q,a), \mathtt{marked}(q,a,b), \mathtt{assign}(i,m)\}$.

3.5.3 Case Study: UJF E-exam

In June 2014, the pharmacy faculty at UJF organized a first e-exam, as a part of *Epreuves Classantes Nationales informatisées* project which aims to realize all medicine exams electronically by 2016. The project is lead by UJF and the e-exam software is developed by the company THEIA specialized in e-formation platforms. This software is currently used by 39 French universities.

We validate our framework by verifying two real e-exam executions passed with this system. All the logs received from the e-exam organizer are anonymized; nevertheless we were not authorized to disclose them. We use MarQ[16] [RCR15] (Monitoring At Runtime with QEA) to model the QEAs and perform the verification. We provide a description for this system that we call *UJF e-exam*, before presenting the results of our analysis.

Exam Description. The UJF exam consist of the following four phases.

Registration: the candidates have to register two weeks before the examination time. Each candidate receives a username/password to authenticate at the examination.

Examination: the exam takes place in a supervised room. Each student handled a previously-calibrated tablet to pass the exam. The internet access is controlled: only IP addresses within an certain range are allowed to access the exam server. A candidate starts by logging in using his username/password. Then, he chooses one of the available exams by entering the exam code, which is provided at the examination time by the invigilator supervising the room. Once the correct code is entered, the exam starts and the first question is displayed. The pedagogical exam conditions mention that the candidates have to answer the questions in a fixed order and cannot get to the next question before answering the current one. A candidate can change the answer as many times as he wants before validating, but once he validates, then he cannot go back and change any of the previously validated answers. Note that, all candidates have to answer the same questions in the same order. A question might be a one-choice question, multiple-choice question, open short-text question, or script-concordance question.

Marking: after the end of the examination phase, the grading process starts. For each question, all the answers provided by the candidates are collected. Then, each answer is evaluated anonymously by an examiner to a mark of 0 if it is wrong, $0 < s < 1$ if it is partially correct, or 1 if it is correct. An example of a partially-correct answer is when a candidate provides only one of the two correct answers for a multiple-choice question. The professor specifies the correct answer(s) and the scores to attribute to correct and partially-correct answers, as well, as the potential penalty. After evaluating all the provided answers for all questions, the total mark for each candidate is calculated as the summation of all the scores attributed to his answers.

Notification: the marks are notified to the candidates. A candidate can consult his submission, obtain the correct answer and his score for each question.

Analysis. We analyzed two exams: Exam 1 involves 233 candidates and contains 42 questions for a duration of 1h35. Exam 2 involves 90 candidates, contains 36 questions for a duration of 5h20. The resulting traces for these exams are respectively of size 1.85 MB and 215 KB and contain 40,875 and 4,641 events. The result of our analysis

[16] https://github.com/selig/qea

together with the time required for MarQ to analyze the whole trace on a standard PC (AMD A10-5745M–Quad-Core 2.1 GHz, 8 GB RAM), are summed up in Table 3.8. (✓) means satisfied, (×) means not satisfied, and (−) indicates the number of candidates that violate the requirements of the property. Note that, some required data are not logged by UJF. Thus, we were not able to verify all our eight properties. Only four of the eight general properties presented in Section 3.5.2.1 were compatible with UJF E-exam. Concerning error reporting properties, MarQ tool did not support all of them. In particular, current version of MarQ tool does not support global variables that are needed of these properties. However, we considered five additional and specific properties for the *UJF exam* which help us to identify the source of error for some properties, and also to detect some candidates that violate the requirements (see below for details).

Property *Candidate Registration* was satisfied, that is, no unregistered candidate submits an answer. *Candidate Eligibility* is also satisfied. We note that, in MarQ tool the *Candidate Eligibility* monitor stops monitoring as soon as a transition to state (2) is made since there is no path to success from state (2). Thus, only the first candidate that violate the requirements is reported. In order to report all such candidates, we had to add an artificial transition from state (2) to an accepting state that could never be taken. Then, monitoring after reaching state (2) remains possible. Moreover, the current implementation of MarQ does not support sets of tuples. Consequently, we could only collect the identities i in a set `F` instead of the tuples (i, q, a).

Answer Authentication was violated only in Exam 1. We reported the violation to the e-exam's developers. The violation actually revealed a discrepancy between the initial specification and the current features of the e-exam software: a candidate can submit the *same answer* several times and this answer remains accepted. Consequently, an event `accept` can appear twice but only with the same answer. To confirm that the failure of *Answer Authentication* is only due to the acceptance of a same answer twice, we update the property *Answer Authentication* and its QEA presented in Figure 3.33 by storing the accepted answer in a variable a_v, and adding a self loop transition on state (3) labeled by $\texttt{accept}(i, q, a)$ $^{[a=a_v]}$. We refer to this new weaker property as *Answer Authentication**, which differs from *Answer Authentication* by allowing the acceptance of the same answer again; but it still forbids the acceptance of a different answer. We found out that *Answer Authentication** is satisfied, which confirms the claim about the possibility of accepting the same answer twice. After diagnosing the source of failure, we defined property *Answer Authentication Reporting* presented in Figure 3.50, which fails if more than one answer (identical or not) is accepted from same candidate to same question. At the same time, it collects all such candidates in a set `F`. *Answer Authentication Reporting* is defined by the QEA depicted in Figure 3.50 with the input alphabet $\Sigma_{\text{AAR}} = \{\texttt{accept}(i, q, a)\}$. The analysis of *Answer Authentication Reporting* shows that, for Exam 1, there is only one candidate such that more than one answer are accepted from him to the same

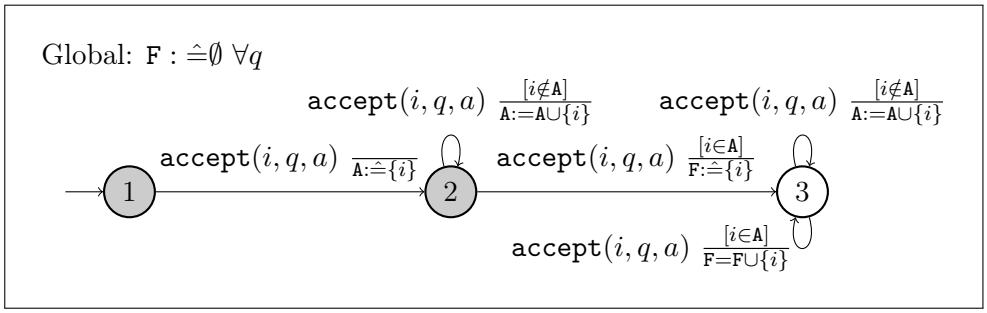


FIGURE 3.50: A QEA for Answer Authentication Reporting.

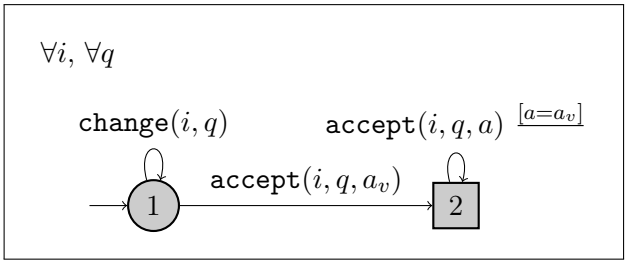


FIGURE 3.51: A QEA for Answer Integrity.

question. The multiple answers that are accepted for the same question are supposed to be equal since *Answer Authentication** is satisfied. Note that MarQ currently does not support global variables, so for *Answer Authentication Reporting*, a set is required for each question. Note that for Exam 1, *Answer Authentication* required less monitoring time than *Answer Authentication** and *Answer Authentication Reporting* as the monitor for *Answer Authentication* stops monitoring as soon as it finds a violation.

Furthermore, *UJF exam* has a requirement stating that after acceptance the writing field is "blocked" and the candidate cannot change it anymore. Actually, in *UJF exam* when a candidate write a potential answer in the writing field the server stores it directly, and once the candidate validates the question the last stored answer is accepted. As *Answer Authentication* shows, several answers can still be accepted after the first acceptance, then the ability of changing the answer in the writing field could result in an acceptance of a different answer. For this purpose, we defined the property *Answer Integrity* that states that a candidate cannot change the answer after acceptance. *Answer Integrity* is defined by the QEA depicted in Figure 3.51 with the input alphabet $\Sigma_{\text{AE}} = \{\texttt{change}(i, q), \texttt{accept}(i, q, a)\}$. Note that, we allowed the acceptance of the same answer to avoid the bug found by *Answer Authentication*. Our analysis showed that *Answer Integrity* was violated in Exam 2: at least one student was able to change the content of the writing field after having his answer accepted.

Concerning *Questions Ordering* the developers did not log anything related to the event $\texttt{get}(i, q)$. However, we defined *Questions Ordering** which fails if a candidate changes the writing field of a future question before an answer for the current question is accepted.

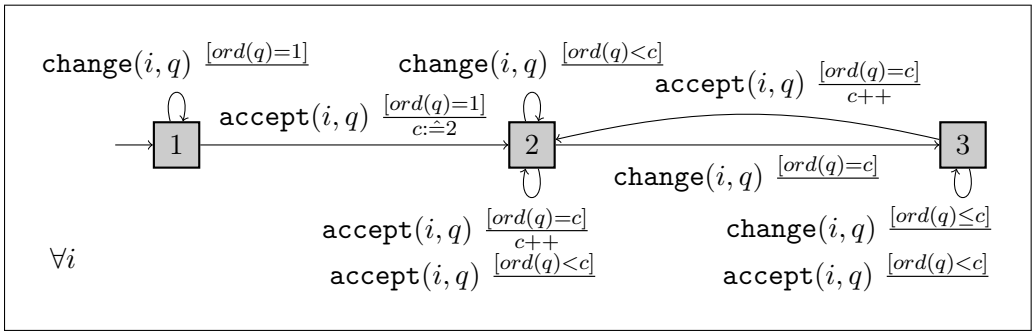


FIGURE 3.52: A QEA for Questions Ordering*.

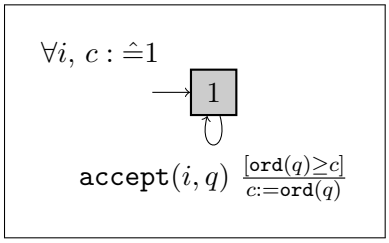


FIGURE 3.53: A QEA for Acceptance Order.

*Questions Ordering** is defined by the QEA depicted in Figure 3.52 with the input alphabet $\Sigma_{\text{QO}'} = \{\texttt{change}(i,q), \texttt{accept}(i,q)\}$. The idea is that if a candidate changes the answer field of a question, he must have received the question previously. Moreover, we allow submitting the same answer twice, and also changing the previous accepted answers to avoid the two bugs previously found. Note that, *UJF exam* requires the candidate to validate the question even if he left it blank, thus we also allow acceptance for the current question before changing its field (self loop above state (2)). The analysis showed that *Questions Ordering** was violated in both exams. Note that, the manual check of *Questions Ordering** showed that the candidates were able to skip certain questions (after writing an answer) without validating them, and then validating the following questions. As we found a violation for *Questions Ordering**, we defined *Acceptance Order* that checks, for each candidate, whether all the accepted answers are accepted in order, *i.e.*, there should be no answer accepted for a question that is followed by an accepted answer for a lower order question. *Acceptance Order* is defined by the QEA depicted in Figure 3.53 with the input alphabet $\Sigma_{\text{AO}} = \{\texttt{accept}(i,q,a)\}$.

Exam Availability is also violated in Exam 2. A candidate was able to change and submit an answer, which is accepted, after the end of the exam duration. We could not analyze *Exam Availability with Flexibility*, since it is not supported by the exam. We also did not consider *Marking Correctness*, and *Mark Integrity* properties since the developers did not log anything concerning the marking and the notification phase is done by each universities and we were not able to get the logs related to this phase. This shows that usually universities only looks for cheating candidates, and do not look for

	Exam 1		Exam 2	
Property	Result	Time (ms)	Result	Time (ms)
Candidate Registration	✓	538	✓	230
Candidate Eligibility	✓	517	✓	214
Answer Authentication	×	310	✓	275
*Answer Authentication**	✓	742	✓	223
Answer Authentication Reporting	×(1)	654	✓	265
Answer Integrity	✓	641	×	218
*Questions Ordering**	×	757	×	389
Acceptance Order	✓	697	✓	294
Exam Availability	✓	518	×(1)	237

TABLE 3.8: Results of the off-line monitoring of two UJF exam runs.

internal problems or insider attacks. We expect the developers of the e-exam software to include logging features for every phase. Note that, we implemented all properties in MarQ and validated them on toy traces as we expect to obtain the actual traces of the marking phase in the near future.

3.6 Conclusion

In this chapter we studied exam protocols. We identify and define several desirable authentication, privacy, verifiability, and monitoring properties. We defined authentication and privacy properties in the Applied π-Calculus, and automatically analyzed using ProVerif the protocols due to Huszti & Pethő [HP10], Giustolisi *et al.* [GLR14], and *Université Grenoble Alpes*. Our analysis shows that Huszti & Pethő protocol indeed satisfies none of the nine authentication and privacy properties. Authentication is compromised because of inaccuracies in the protocol design, whereas most of attacks invalidating privacy exploit attacks on the RARC. These attacks compromise secrecy and anonymity of the messages, and exploit the absence of a proof of knowledge of the submitted message to the RARC, which allows its use as a decryption oracle. Such a proof is not explicitly required in the original specification of the RARC, and is certainly missing in the H&P protocol: the "exam authority" is required to forward questions and answers without knowing them, and thus cannot prove knowledge of them when submitting them to the RARC. Even when assuming a perfect RARC ensuring anonymity, we still have attacks on all properties. Thus, we think that fixing the RARC is not sufficient – the protocol requires fundamental changes. Remark! protocol presents a weakness concerning *Form Authenticity*. We propose a fix and formally verify that the (fixed) protocol satisfies all the properties herein considered. *Université Grenoble Alpes* exam fails to satisfy *Anonymous Examiner* and *Mark Privacy*.

In the second part of the chapter, we proposed general abstract definitions of exam Verifiability properties. For each property, we define the soundness and completeness conditions for the related verification test. Then, we instantiated our properties and analyzed with the help of ProVerif the protocols due to Giustolisi *et al.* [GLR14], and *Université Grenoble Alpes*. The analysis of Giustolisi *et al.* shows that all properties but three are satisfied without assuming that the exam's roles are honest. However, *Marking Correctness* holds only assuming an honest exam authority. In fact, a student can check her mark by using the exam table, but this is posted on the bulletin board by the exam authority who can nullify the verification of correctness by tampering with the table. Whereas the analysis of *Université Grenoble Alpes* exam shows that it satisfies all the verifiability properties under the assumption that authorities and examiner are honest. This seems to be peculiar to paper-and-pencil exams, where log-books and registers are managed by the authorities that can tamper with them. Only *Marking Correctness* holds even in presence of dishonest authorities and examiner: here, a candidate can consult her exam-test after marking, thus verifying herself whether her mark has been computed correctly.

In the final part, we defined monitors that allow us to check the exam requirements at runtime. Then, we implemented these monitors and analyzed an e-exam organized by *Université Joseph Fourier* using MarQ tool. The analysis reveals both fraudulent students and discrepancies between the specification and the implementation. Note that, due to the lack of logs about the marking and notification phases, we were not able to analyze all properties. The *UJF E-exam* case study clearly demonstrates that the developers do not think to log these two phases where there is less interaction with the candidates. However, we believe that monitoring the marking phase is essential since a successful attempt from a bribed examiner or a cheating student can be very effective.

As a future work, it would be interesting to model the full Huszti & Pethő protocol with RARC as we analyzed RARC alone due to ProVerif termination problem. Also, it would be great to automatize the manual proofs used to prove the general case of universal verifiability properties. Another line of research is to study novel properties such as accountability, which allows them to identify which party is responsible when the verification fails. Few works go in this direction: Küsters *et al.* [KTV10] have studied accountability, however, their framework needs to be instantiated for each application by identifying relevant verifiability goals. As we identified several verifiability properties relevant for exam protocols, one may study how their accountability framework can be applied to case of exam protocols. Bella *et al.* [BGLR15] have proposed an accountability property, which allows to identify the responsible party when a candidate fails to submit an answer or to receive the corresponding mark. They have analyzed their protocol and have shown that it satisfies this property. However, more accountability properties need to be defined to identify the responsible parties when the verifiability properties

we propose in this thesis fails. With respect to runtime verification, one direction is to perform the verification of marking related properties, as well as, to perform online verification with our monitors during live e-exams, and to study to what extent runtime enforcement can be applied during a live e-exam run. Online verification requires to deliver events to the monitor at the time they are generated by the system in order to check them and take a verdict. This introduces additional overheads and may cause scalability problems depending on the size of the system and number of events generated *per* second. Another direction is to study more expressive and quantitative properties that might detect possible collusions between students through similar answer patterns.

Chapter 4

e-Cash Protocols

Electronic cash (e-cash) aims at achieving client privacy during payment, similar to real cash. Due to digital nature of electronic coins security against abuse of individuals is also a main concern to prevent generating fake coins, or spending the same coin twice. In this chapter, we propose a formal framework to define, and verify security properties of e-cash protocols. We define two client privacy properties and three properties to prevent forgery. Then, we apply our definitions and analyze using ProVerif the Chaum protocol [Cha82], the DigiCash protocol[1], and the Chaum *et al.* protocol [CFN88].

Contents

4.1 Introduction

Although current banking and electronic payment systems such as credit cards or, *e.g.*, PayPal allow clients to transfer money around the world in a fraction of a second, they

[1] DigiCash Inc. was an electronic money corporation founded by David Chaum in 1990. The protocol used by DigiCash has been presented by Berry Schoenmakers in [Sch97].

do not fully ensure the clients' privacy. In such systems, no transaction can be made in a completely anonymous way, since the bank or the payment provider knows the details of the clients' transactions. By analyzing a client payments for, *e.g.*, transportations, hotels, restaurants, movies, clothes, and so on, the payment provider can typically deduce the client's whereabouts, and much information about his lifestyle.

Physical cash provides better privacy: the payments are difficult to trace as there is no central authority that monitors all transactions, in contrast to most electronic payment systems. This property is the inspiration for "untraceable" e-cash systems. The first such e-cash system preserving the client's anonymity was presented by David Chaum [Cha82]. A client can withdraw a coin anonymously from his bank and spend it with a seller. The seller can then deposit the coin at the bank, who will credit his account. In this protocol coins are *non-transferable*, *i.e.*, the seller cannot spend a received coin again, but has to deposit it at the bank. If he wants to spend a coin in another shop, he has to withdraw a new coin from his account, similar to the usual payment using cheques. In contrast, there are protocols where coins are also *transferable*, *i.e.*, coins do not need to be deposited directly after each spend, but can be used again, *e.g.*, [OO89, CGT08].

To be secure, an e-cash protocol should not only ensure the client's privacy, but must also ensure that a client cannot *forge* coins which were not issued by the bank. Moreover, it must protect against *double spending* – otherwise a client may use the same coin multiple times. This can be achieved by using online payments, *i.e.*, a seller has to contact the bank at payment before accepting the coin, however it is an expensive solution. An alternative solution, which is usually used to support offline payments (*i.e.*, a seller can accept the payment without contacting the bank), is to reveal the client's identity if he spends a coin twice. Note that to avoid double spending, systems like Bitcoin [Nak08] uses a decentralized approach, where a consensus among users substitutes the bank. So, when a transaction is made it exposure to double spending with less and less risk as it gains confirmations. Finally, *exculpability* ensures that an attacker cannot forge a double spend, and hence incorrectly blame an honest client for double spending.

Contributions. We provide in this chapter the following contributions:

- We propose a formal framework to verify security properties of non-transferable e-cash protocols. We model e-cash systems in the Applied π-Calculus [AF01]. Then, we define two client privacy properties and three properties to prevent forgery.
- We illustrate our model by analyzing using ProVerif [Bla01] three case studies: the Chaum protocol [Cha82], the DigiCash protocol, and the Chaum *et al.* protocol [CFN88].

Outline of the Chapter. In Section 4.2, we discuss the related work. We formally model e-cash protocol in Applied π-Calculus in Section 4.3. Then, we define the forgery

related properties in Section 4.4.1 and the privacy properties in Section 4.4.2. In Sections 4.5.1, 4.5.2, and 4.5.3, we respectively analyze the protocols due to Chaum [Cha82], DigiCash [Sch97], and Chaum *et al.* [CFN88]. Finally, we conclude in Section 4.6.

4.2 Related Work

In the literature, several e-cash protocols have been proposed [Cha82, CFN88, Dam88, DC94, Cre94, Bra93, AF96, KO02, FHY13]. For example, Chaum [Cha82] has proposed an online e-cash protocol based on blind signature. Latter, Chaum *et al.* [CFN88] presented an offline variant of this protocol. Berry Schoenmakers has described a real e-cash protocol that is implemented by DigiCash based on online Chaum protocol [Cha82]. Abe *et al.* [AF96] have introduced a scheme based on partial blind signature, which allows the signer (the bank) to include certain information in the blind signature of the coin, for example the expiration date or the value of the coin. Kim *et al.* [KO02] have proposed an e-cash system that supports coin refund and assigns them a value, based again on partial blind signature.

At the same time, several attacks have been found against various existing e-cash protocols: for example Pfitzmann *et al.* [PW91, PSW95] break the anonymity of [Dam88, DC94, Cre94]. Cheng *et al.* [CYS05] show that Brand's protocol [Bra93] allows a client to spend a coin more than once without being identified. Aboud and Agoun [AA14] show that [FHY13] cannot ensure the anonymity and unlinkability properties that were claimed. These numerous attacks triggered some first work on formal analysis of e-cash protocols in the computational [CG08] and symbolic world [LCPD07, SK14]. Canard and Gouget [CG08] provide formal definitions for various privacy and unforgeability properties in the computational world, but only with manual proofs as their framework is difficult to automate. In contrast, Luo *et al.* [LCPD07] and Thandar *et al.* [SK14] both rely on automatic tools (ProVerif [Bla01], and AVISPA [ABB+05] respectively). Yet, they only consider a fraction of the essential security properties, and for some properties Luo *et al.* only perform a manual analysis. Moreover, much of their reasoning is targeted on their respective case studies, and cannot easily be transferred to other protocols.

4.3 Modeling E-cash Protocols

In this section, we model e-cash protocol in the Applied π-Calculus. An e-cash system involves the following parties: the *client* C who has an account at the bank, the *seller* S who accepts electronic coins, and the bank B, which certifies the electronic coins. E-cash protocols typically run in three phases:

1. **Withdrawal:** the client withdraws an electronic coin from the bank, which debits the client's account.
2. **Payment:** the client spends the coin by executing a transaction with a seller.
3. **Deposit:** the seller deposits the transaction at the bank, which credits the seller's account.

In addition to these three main phases, some systems allow the clients:

(a) to return coins directly to the bank without using them in a payment, for instance in case of expiration, or to re-distribute the coins denominations, and

(b) to restore coins that have been lost, for instance due to a hard disk crash.

As these functionalities are not implemented by all protocols, our model does not require them. Moreover, we assume that the coins are neither transferable nor divisible. We define an e-cash protocol as a tuple of processes each representing the role of a certain party.

Definition 4.1. (E-cash Protocol). *An* e-cash protocol *is a tuple* $(B, S, C, \tilde{n}_p)$*, where* B *is the process executed by the bank,* S *is the process executed by the sellers,* C *is the process executed by the clients, and* $\tilde{n}_p$ *is the set of the private channel names used by the protocol.*

To reason about privacy properties, we use concrete instances of an e-cash protocol, called *e-cash processes.*

Definition 4.2. (E-cash Process). *Given an e-cash protocol, an* e-cash process *is a closed plain process:*

$$\begin{aligned} CP \equiv \nu\tilde{n}.(B|S\sigma_{ids_1}|\ldots|S\sigma_{ids_l}| \\ (C\sigma_{idc_1}\sigma_{c_{11}}\sigma_{ids_{11}}|\ldots|C\sigma_{idc_1}\sigma_{c_{1p_1}}\sigma_{ids_{1p_1}})| \\ \vdots \\ |(C\sigma_{idc_k}\sigma_{c_{k1}}\sigma_{ids_{k1}}|\ldots|C\sigma_{idc_k}\sigma_{c_{kp_k}}\sigma_{ids_{kp_k}})) \end{aligned}$$

where $\tilde{n}$ *is the set of all restricted names which includes some of the protocol's private channels* $\tilde{n}_p$*;* B *is the process executed by the bank;* $S\sigma_{ids_i}$ *is the process executed by the seller whose identity is specified by the substitution* σ_{ids_i}*;* $C\sigma_{idc_i}\sigma_{c_{ij}}\sigma_{ids_{ij}}$ *is the process executed by the client whose identity is specified by the substitution* σ_{idc_i}*, and which spends the coin identified by the substitution* $\sigma_{c_{ij}}$ *to pay the seller with the identity specified by the substitution* $\sigma_{ids_{ij}}$*. Note that* idc_i *can spend* p_i *coins.*

Similar to previous chapter, we introduce the notation of *e-cash context* in order to improve the readability of our definitions. An e-cash context $CP_I[\_]$ is an e-cash process CP with "holes" for all processes executed by the parties whose identities are included in a set I. For example, to enumerate all the sessions executed by the Client idc_1 without repeating the entire e-cash instance, we can rewrite CP as $CP_{\{idc_1\}}[C\sigma_{idc_1}\sigma_{c_{11}}\sigma_{ids_{11}}|\ldots|C\sigma_{idc_1}\sigma_{c_{1p_1}}\sigma_{ids_{1p_1}}]$.

Definition 4.3. (E-cash Context). *Let I be a set such that $I \subseteq I_S \cup I_C \cup \{id_B\}$ where I_S is the set of all sellers, I_C is the set of all clients, id_B identifies the bank B. Then, given an e-cash process*

$$\begin{aligned} CP \equiv \nu\tilde{n}.(B|S\sigma_{ids_1}|\ldots|S\sigma_{ids_l}| \\ (C\sigma_{idc_1}\sigma_{c_{11}}\sigma_{ids_{11}}|\ldots|C\sigma_{idc_1}\sigma_{c_{1p_1}}\sigma_{ids_{1p_1}})| \\ \vdots \\ |(C\sigma_{idc_k}\sigma_{c_{k1}}\sigma_{ids_{k1}}|\ldots|C\sigma_{idc_k}\sigma_{c_{kp_k}}\sigma_{ids_{kp_k}})) \end{aligned}$$

we define e-cash context $CP_I[\_]$ *as follows:*

$$CP_I[\_] \equiv \nu\tilde{n}.\Big((B)^k|\underset{ids_i\notin I}{S\sigma_{ids_i}}|\_|\underset{idc_i\notin I}{(C\sigma_{idc_i}\sigma_{c_{i1}}\sigma_{ids_{i1}}|\ldots|C\sigma_{idc_i}\sigma_{c_{ip_i}}\sigma_{ids_{ip_i}})}\Big)$$

where $k = 0$ if $id_B \in I$, and $k = 1$ otherwise.

We also use the notation C_w to denote a client that withdraws a coin, but does not spend it in a payment: C_w is a variant of the process C that halts at the end of withdrawal phase, i.e., where the code corresponding to the payment phase is removed. Note, only honest parties are modeled in an e-cash process. Honest parties are those that follow the protocol's specification, and particularly neither reveal their secret data (*e.g.*, account numbers, keys etc.) to the attacker, nor take malicious actions such as double spending a coin or generating fake transactions. In addition to the attacker, we consider corrupted parties, who communicate with the attacker, share personal data with him, or receive orders from him. Again as in exams, we model a corrupted party as P^{ch_1,ch_2} which is introduced in the Definition 2.2 in Section 2.1.1.

To define forgery related properties, we use the following events, sets, and function:

- `withdraw(c)`: is an event emitted when the coin c is withdrawn. This event is placed inside the bank process just after the bank outputs the coin's certificate (*e.g.*, a signature on the coin).
- `spend(c)`: is an event emitted when the coin c is spent. This event is placed inside the seller process just after he receives and accepts the coin.

- `TR`: is the set of all possible transactions.
- `ID`: is the set of all client identities.
- `D`: is a special data set that includes the data known to the bank after the protocol execution, *e.g.*, the data presents in the bank's database.
- `transId`: is a function that takes a transaction $tr \in \texttt{TR}$ and returns a pair (s, c), where s identifies tr and c is the coin involved in tr. Such a pair can usually be computed from a transaction.

4.4 Security Properties

We first define three forgery related properties: *Unforgeability*, *Double Spending Identification*, and *Exculpability*, before defining two privacy properties: *Weak Anonymity* and *Strong Anonymity*.

4.4.1 Forgery-Related Properties

In an e-cash protocol a client must not be able to create a coin without involving the bank, resulting in a fake coin, or to double spend a valid coin he withdrew from the bank. This is ensured by *Unforgeability*, which says that the clients cannot spend more coins than they withdrew.

Definition 4.4. (Unforgeability). *An e-cash protocol ensures* Unforgeability *if for every e-cash process CP on every possible execution trace, each occurrence of the event* `spend(c)` *is preceded by a distinct occurrence of the event* `withdraw(c)`.

If a fake coin is successfully spent, the event `spend` will be emitted without any matching event `withdraw`, violating the property. Similarly, in the case of a successful double spending the event `spend` will be emitted twice, but these events are preceded by only one occurrence of the event `withdraw`. In the rest of this chapter, we illustrate all our notions with the "*real cash*" system (mainly coins and banknotes) as a running example. We hope that it helps the reader to understand the properties but also to feel the difference between real cash and e-cash systems.

Example 4.1. (Unforgeability in Real Cash). *In real cash, unforgeability is ensured by physical measures that make forging or copying coins and banknotes difficult, for example by adding serial numbers, using ultraviolet ink, holograms and so on.*

Since a malicious client might be interested to create fake coins or double spend a coin, it is particularly interesting to study *Unforgeability* with an honest bank and corrupted clients. A partially corrupted seller, which *e.g.*, gives some information to the attacker

but still emits the event `spend` correctly, could also be considered. This allows us to check if a seller colluding with the client and the attacker can result in a coin forging. Note that, if the seller is totally corrupted then *Unforgeability* can be trivially violated, since a corrupted seller can simply emit the event `spend` for a forged coin, although there was no transaction.

In case of double spending, the bank should be able to identify the responsible client. This is ensured by *Double Spending Identification* (DSI), which says that a client cannot double spend a coin without revealing his identity. To deposit a coin at the bank the seller has to present a *transaction* which contains, in addition to the coin, some information certifying that he received the coin in a payment. A *valid transaction* is a transaction which could be accepted by the bank, *i.e.*, it contains a correct proof that the coin is received in a correct payment. The bank accepts a valid transaction if it does not contain a coin that is already deposited using the same or a different transaction.

Definition 4.5. (Double Spending Identification). *An e-cash protocol ensures* Double Spending Identification *if there exists a test* $\mathtt{T_{DSI}} : \mathtt{TR} \times \mathtt{TR} \times \mathtt{D} \mapsto \mathtt{ID} \cup \{\bot\}$ *satisfying: for any two valid transactions* tr_1 *and* tr_2 *that are different but involve the same coin (i.e.,* $\mathtt{transId}(tr_1) = (s_1, c)$*, and* $\mathtt{transId}(tr_2) = (s_2, c)$ *for some coin* c *with* $s_1 \neq s_2$*), there exists* $p \in \mathtt{D}$ *such that* $\mathtt{T_{DSI}}(tr_1, tr_2, p)$ *outputs* $(idc, e) \in \mathtt{ID} \times \mathtt{D}$*, where* e *is an evidence that* idc *withdrew the coin* c*.*

Double Spending Identification allows the bank to identify the double spender by running a test $\mathtt{T_{DSI}}$ on two different transactions that involve the same coin. For example, consider a protocol where after a successful transaction the seller gets $x = m.id + r$ where id is the identity of the client (*e.g.*, his secret key), r is a random value (identifies the coin) chosen by the client at withdrawal, and m is the challenge of the seller. So, if the client double spends the same coin then the bank can compute id and r using the two equations: $x_1 = m_1.id + r$ and $x_2 = m_2.id + r$. The data p could be some information necessary to identify the double spender or to construct the evidence e. This data is usually presented to the bank at withdrawal or at deposit. The required evidence depends on the protocol. Note that e is an evidence from the point of view of the bank, and not necessarily a proof for an outer judge. Thus, the goal of *Double Spending Identification* is to preserve the security of the bank so that he can detect and identify the responsible of double spending when this event occurs. Note that, if a client withdraws a coin and gives it to an attacker which double spends it, then the test returns the identity of the client and not the attacker's identity.

Example 4.2. (DSI in Real cash). *In real cash, double spending is prevented by ensuring that notes cannot be copied. However,* Double Spending Identification *is not ensured: even if a central bank is able to identify copied banknotes using, e.g., their*

serial numbers, this does not allow it to identify the person responsible for creating the counterfeit notes.

Double Spending Identification gives rise to a potential problem: what if the client is honest and spends the coin only once, but the attacker (*e.g.*, a corrupted seller) is able to forge a second spend, or what if a corrupted bank is able to simulate a coin withdrawal and payment *i.e.*, to forge a coin withdrawal and payment that seems to be made by a certain client. For instance, in the example mentioned above, the two equations are enough evidence for the bank. However, if the bank knows *id* he can generate the two equations himself and blame the client for double spending. So, to convince a judge, an additional evidence is needed, *e.g.*, the client's signature. If any of the two situations mentioned above is possible, then a honest client could be falsely blamed for double spending, and also it gives raise to a corrupted client which is responsible of double spending to deny it. To solve this problem we define *Exculpability*, which says that the attacker, even when colluding with the bank and the seller, cannot forge a double spend by a honest client in order to blame him. More precisely, provided a transaction executed by a client *idc*, the attacker cannot provide two different valid transactions which involves the same coin, and the data p necessary for the test $\mathtt{T}_{\mathtt{DSI}}$ to output the identity *idc* with an evidence. Note that *Exculpability* is only relevant if *Double Spending Identification* holds: otherwise a client cannot be blamed regardless of the ability to forge a second spend or to simulate a coin withdrawal and payment, as his identity cannot be revealed.

Definition 4.6. (Exculpability). *Assume that we have a test* $\mathtt{T}_{\mathtt{DSI}}$ *as specified in Definition 4.5, i.e.,* Double Spending Identification *holds, and that the bank is corrupted. Let idc be a honest client (in particular he does not double spend a coin), and ids be a corrupted seller. Then,* Exculpability *is ensured if, after observing a transaction made by idc with ids, the attacker cannot provide two valid transactions* $tr_1, tr_2 \in \mathtt{TR}$ *that are different but involve the same coin c, and some data p such that* $\mathtt{T}_{\mathtt{DSI}}(tr_1, tr_2, p)$ *outputs* (idc, e) *where e is an evidence that idc withdrew the coin c.*

The intuition is: if the attacker can provide two transactions tr_1, tr_2 such that $\mathtt{T}_{\mathtt{DSI}}(tr_1, tr_2)$ returns a client's identity (that is, two different valid transactions involve the same coin), then it was able to forge (at least) one transaction since the honest client performs (at most) one transaction *per* coin. If after observing a transaction executed by a client *idc*, the attacker can provide a different valid transaction which involves the same coin, and the required data p, then the test returns the identity *idc* with the necessary evidence, thus the property will be violated. Similarly, in the case where the attacker can forge a coin withdrawal and payment seems to be made by a client *idc*, then the attacker can obtain two transactions satisfying the required conditions, together with the necessary data p, so that the test returns the identity *idc* with an evidence. Note that, *Double*

Spending Identification and *Exculpability* are only relevant in case of offline e-cash systems where double spending is possible.

4.4.2 Privacy Properties

We express our privacy properties as observational equivalence. We use the *labeled bisimilarity* ($\approx_l$) to express the equivalence between two processes [AF01]. To ensure the privacy of the client, the following two notions have been introduced by cryptographers and are standard in the literature *e.g.*, [CG08, Fer93, Sch97].

1. *Weak Anonymity*: the attacker cannot link a client to a spend, *i.e.*, he cannot distinguish which client makes the payment.

2. *Strong Anonymity*: additionally to weak anonymity, the attacker should not be able to decide if two spends were done by the same client, or not.

Canard and Gouget [CG08] define *Weak Anonymity* as the following. Two honest clients each withdraw a coin from the bank. Then one of them (randomly chosen) spends his coin to the adversary. The adversary already knows the identities of these two clients, and also the secret key of the bank. It wins the game if it guesses correctly which client spends the coin. Inspired by this definition, we define *Weak Anonymity* as follows.

Definition 4.7. (Weak Anonymity). *An e-cash protocol ensures* Weak Anonymity *if for any e-cash process CP, any corrupted Seller ids, and any two honest Clients idc_1 and idc_2, we have that:*

$$CP_I[C\sigma_{idc_1}\sigma_{c_1}\sigma_{ids} | C_w\sigma_{idc_2}\sigma_{c_2} | (S\sigma_{ids})^{ch_1,ch_2} | B^{ch_1,ch_2}]$$
$$\approx_l$$
$$CP_I[C_w\sigma_{idc_1}\sigma_{c_1} | C\sigma_{idc_2}\sigma_{c_2}\sigma_{ids} | (S\sigma_{ids})^{ch_1,ch_2} | B^{ch_1,ch_2}]$$

where c_1, c_2 are any two coins (not previously known to the attacker) withdrawn by idc_1 and idc_2 respectively, $I = \{idc_1, idc_2, ids, id_B\}$, id_B is the bank's identity, and C_w is a variant of C that halts at the end of the withdrawal phase.

Weak anonymity ensures that a process in which the client idc_1 spends the coin c_1 to the corrupted seller ids_1, is equivalent to a process in which the client idc_2 spends the coin c_2 to the corrupted seller ids_1. We assume a corrupted bank represented by B^{ch_1,ch_2}. Note that the client that does not spend his coin still withdraws it. This is necessary since otherwise the attacker could likely distinguish both sides during the withdrawal phase, as the bank is corrupted and typically the client reveals his identity to the bank at withdrawal. We also note that we do not necessarily consider other corrupted clients, however this can easily be done by replacing some honest clients from the context CP_I (*i.e.*, other than idc_1 and idc_2) with corrupted ones.

Example 4.3. (Weak anonymity in Real cash). *Real coins ensure weak anonymity as two coins (assuming the same value and production year) are indistinguishable. However, banknotes do not ensure weak anonymity according to our definition, as they include serial numbers. Since the two clients withdraw a note each, the notes hence have different serial numbers which the bank can identify. In reality this is used by central banks to trace notes and detect suspicious activities that, e.g., could hint at money laundering. Note however that banknotes ensure a weaker form of anonymity: if two different clients use the same note, one cannot distinguish them.*

A stronger privacy property is *Strong Anonymity*, which is defined in [CG08] using a similar game as *Weak Anonymity* with the difference that the adversary may have previously seen some coins being spent by the two honest clients explicitly mentioned in the definition. Such definition of anonymity resembles untraceability. In symbolic model, we define *Strong Anonymity* as follows:

Definition 4.8. (Strong Anonymity). *An e-cash protocol ensures* Strong Anonymity *if for any e-cash process CP, any corrupted seller ids, and any two honest clients idc_1 and idc_2, we have that:*

$$CP_I[\ |\ \underset{0\le i\le m_1}{C\sigma_{idc_1}\sigma_{c_1^i}\sigma_{ids}}\ |\ \underset{0\le i\le m_2}{C\sigma_{idc_2}\sigma_{c_2^i}\sigma_{ids}}|C\sigma_{idc_1}\sigma_{c_1}\sigma_{ids}|C_w\sigma_{idc_2}\sigma_{c_2}|(S\sigma_{ids})^{ch_1,ch_2}|B^{ch_1,ch_2}]$$

$$\approx_l$$

$$CP_I[\ |\ \underset{0\le i\le m_1}{C\sigma_{idc_1}\sigma_{c_1^i}\sigma_{ids}}\ |\ \underset{0\le i\le m_2}{C\sigma_{idc_2}\sigma_{c_2^i}\sigma_{ids}}|C_w\sigma_{idc_1}\sigma_{c_1}|C\sigma_{idc_2}\sigma_{c_2}\sigma_{ids}|(S\sigma_{ids})^{ch_1,ch_2}|B^{ch_1,ch_2}]$$

where c_1 and $c_1^1 \dots c_1^{m_1}$ are any coins withdrawn by idc_1, c_2 and $c_2^1 \dots c_2^{m_2}$ are any coins withdrawn by idc_2, $I = \{idc_1, idc_2, ids, id_B\}$, id_B is the bank's identity, and C_w is a variant of C that halts at the end of the withdrawal phase.

Strong Anonymity ensures that the process in which the client idc_1 spends $m_1 + 1$ coins, while idc_2 spends m_2 coins and additionally withdraws another coin without spending it, is equivalent to the process in which the client idc_1 spends m_1 coins and withdraws an additional coin, while idc_2 spends $m_2 + 1$ coins. The definition assumes that the bank is corrupted, and that the seller receiving the coins from the two clients idc_1 and idc_2 is also corrupted. Note that, we consider C_w to avoid distinguishing from the number of withdrawals by each client. Again, we can replace some honest clients from CP_I by corrupted ones.

Example 4.4. (Strong Anonymity in Real cash). *Again, real coins ensure strong anonymity as, assuming the same value and production year, two coins are indistinguishable. Yet, for the same reason as in weak anonymity, banknotes do not ensure strong anonymity according to our definition: the serial numbers allow an attacker to identify the different clients.*

We note that any protocol satisfying *Strong Anonymity* also satisfies *Weak Anonymity*, as *Weak Anonymity* is a special case of Strong Anonymity for $m_1 = m_2 = 0$, i.e. when the two honest clients do not make any previous spends.

4.5 Case Studies

David Chaum proposed an online e-cash system in [Cha82] based on blind signature. An offline variant of this protocol is proposed by Chaum *et al.* in [CFN88]. A real implementation based on Chaum protocol (in its two variants), which allows to make purchases over open networks such as the Internet, put in service by the DigiCash Inc. which was founded by David Chaum in 1990. The corporation declared bankruptcy later in 1998, and was sold to Blucora[2] (formerly Infospace Inc.). The online protocol implemented by DigiCash is presented by Berry Schoenmakers in [Sch97]. In this section, we describe and analyze both the Chaum (online) protocol [Cha82] and the Chaum *et al.* (offline) protocol [CFN88], as well as, the online protocol implemented by DigiCash [Sch97] (DigiCash protocol). For this we use ProVerif an automatic tool that verifies cryptographic protocols. Note that, all the verification presents in this section are carried out on a standard PC (Intel(R) Pentium(R) D CPU 3.00GHz RAM 2GB).

4.5.1 Chaum Protocol

The Chaum Protocol is proposed in [Cha82] and summarized in [CFN88]. It allows a client to withdraw a coin blindly from the bank, and then spend it later in a payment without being traced even by the bank. We start by given a description of the protocol, then we show how we model it in ProVerif and discuss the obtained results.

Description. The Chaum Protocol is an online protocol in the sense that the seller does not accept the payment before contacting the bank to verify that the coin is not deposited before, this is to prevent double spending. The Chaum Protocol is composed of the following phases:

Withdrawal Phase. To obtain an electronic coin, the client communicates with the bank using the following protocol:

1. The client randomly chooses a value x, and a coefficient r, the client before sending to the bank his identity u and the value $b = \texttt{blind}(x, r)$, where `blind` is a blinding function.

2. The bank signs the blinded value b using a signing function `sign` and his secret key skB, before sending the signature $bs = \texttt{sign}(b, skB)$ to the client. The bank also debits the amount of the coin from the client's account.

[2] http://www.blucora.com/

3. The client verifies the signature and removes the blinding to obtain the bank's signature $s = \texttt{sign}(x, skB)$ on x. The coin consists of the pair $(x, \texttt{sign}(x, sk_B))$.

Payment (*and deposit*) *Phases.* To spend a coin the following steps are taken:

1. The client sends the pair $(x, \texttt{sign}(x, sk_B))$ to the seller.
2. After checking the bank's signature, the seller sends the coin $(x, \texttt{sign}(x, sk_B))$ to the bank to verify that it is not deposited before.
3. The bank verifies the signature, and that the coin is not in the list of deposited coins. If these checks succeed the bank credits the seller's account with the amount of the coin and informs him of acceptance. Otherwise, the payment is rejected.

Modeling in ProVerif. We use ProVerif to perform the automatic protocol verification. We model privacy properties as equivalence properties, and we use events to verify the other properties. The equational theory depicted in Figure 4.1 models the cryptographic primitives used within Chaum protocol. It includes well-known model for digital signature (functions `sign`, `getmess`, and `checksign`). The functions `blind`/`unblind` are used to blind/unblind a message using a random value. We also include the possibility of unblinding a signed blinded message, so that we obtain the signature of the message – the key feature of blind signatures.

$$\begin{aligned} \texttt{getmess}(\texttt{sign}(m, k)) &= m \\ \texttt{checksign}(\texttt{sign}(m, k), pk(k)) &= m \\ \texttt{unblind}(\texttt{blind}(m, r), r) &= m \\ \texttt{unblind}(\texttt{sign}(\texttt{blind}(m, r), k), r) &= \texttt{sign}(m, k) \end{aligned}$$

FIGURE 4.1: Equational theory for our model of Chaum protocol.

Analysis. The result of the analysis is summarized in Table 4.1. We model *Unforgeability* as an injective correspondence between the two events `withdraw` and `spend`, they are placed in their appropriate position, according to the Definition 4.4, inside the bank and seller processes respectively. We consider a honest bank and a honest seller but corrupted clients. We assume that the bank sends an authenticated message through private channel to inform the seller about a coin acceptance. Otherwise, the attacker can forge a message which results in an acceptence of already deposited coin. However, ProVerif still found an attack against *Unforgeability* that is when two copies of the same coin spent at the same time. In this case the bank makes two parallel database lookup to check if the coin is deposited before. Each lookup confirms that the coin is not deposited before if the parallel lookup not finished yet and thus the coin is not yet inserted in the database.

Property	Result	Time
Unforgeability	×	$< 1s$
Weak Anonymity	✓	$< 1s$
Strong Anonymity	✓	$< 1s$

TABLE 4.1: Results of our analysis on the formal model of the Chaum protocol. (✓) indicates that the property holds. (×) indicates that it fails (ProVerif shows an attack).

This results in an acceptance of two spends of the same coin. The latter attack may be avoided with some synchronization like locking the table when a coin deposit is initiated and then unlock it when the operation is finished. However, such a locking mechanism may create deadlocks in practice. ProVerif does not support such an feature. Protocols that rely on databases can be analyzed using the Tamarin Prover [MSCB13] thanks to the SAPIC[3] [KK14] tool. Note that, corrupted client cannot create a fake coin as the correspondence holds without injectivity. In case of online protocols, *Double Spending Identification* and *Exculpability* are not relevant as online protocols tackle double spending by calling of the bank at payment. Thus, they do not have any kind of test to identify double spender.

For privacy properties, we assume corrupted bank and corrupted seller, but honest client. ProVerif confirms that the privacy of the client is preserved, as both *Weak Anonymity*, and *Strong Anonymity* are satisfied. This due to the fact that the coin is signed blindly during the withdrawal phase, and thus cannot be traced later by the attacker even when colludes with the bank and the seller. Note that, we have to perform some "shuffling" of the coins at payment in order to verify privacy properties using ProVerif[4]. Note also that, for *Strong Anonymity* we consider unbounded number of spends by each client with one spend that is made by either the first client or by the second one.

4.5.2 DigiCash Protocol

Chaum protocol is implemented by DigiCash Inc. We analyze the specification the implemented protocol as presented by Berry Schoenmakers in [Sch97]. In the following, we give the description of the protocol, our modeling in Proverif, and the result of our analysis.

Description. The DigiCash protocol has the same withdrawal phase as Chaum protocol, except that the client sends an authenticated coin to the bank in order to sign it. However, the paper [Sch97] does not specify the way of authentication. We ignore this authentication as its purpose is to ensure that the bank debits the correct client account. Hence, we believe that it does not effect the privacy and unforgeability properties (analysis confirms

[3] http://sapic.gforge.inria.fr/ [4] Similar, for using $out(a, \texttt{choice}[m, n]) \mid out(a, \texttt{choice}[n, m])$ instead of $out(a, \texttt{choice}[m, m]) \mid out(a, \texttt{choice}[n, n])$ to prove observational equivalence.

Property	Result	Time
Unforgeability	×	$< 1s$
Weak Anonymity	✓	$< 1s$
Strong Anonymity	✓	$< 1s$

TABLE 4.2: Results of our analysis on the formal model of the DigiCash protocol. (✓) indicates that the property holds. (×) indicates that it fails (ProVerif shows an attack).

that as we can see in Table 4.1). The payment and deposit phases are different from those of Chaum protocol. They are summarized by the following steps:

1. The client sends to the seller $pay = enc((id_s, \mathtt{h}(\textit{pay-spec}), x, \mathtt{sign}(x, sk_B)), pk_B)$ which is the encryption, using the public key of the bank pk_B, of the seller's identity id_s, hash of the payment specification *pay-spec* (specification of the sold object, price etc), and the coin $(x, \mathtt{sign}(x, sk_B))$.

2. The seller signs $(\mathtt{h}(\textit{pay-spec}), pay)$ and sends it with his identity id_s to the bank.

3. The bank verifies the signature, decrypts *pay* then verifies the value of $\mathtt{h}(\textit{pay-spec})$ and that the coin is valid and not deposited before. If so it informs the seller to accept the coin, and to reject it otherwise.

Modeling in ProVerif. Additionally to the equational theory of the Chaum protocol (Table 4.1), the equational theory of DigiCash protocol includes well-known model of the public key encryption represented by the following equation: $\mathtt{dec}(\mathtt{enc}(m, \mathtt{pk}(k)), k) = m$, where $\mathtt{pk}(k)$ represents the public key corresponding to the key k.

Analysis. The result of analysis of DigiCash protocol using ProVerif is summarized in Table 4.2. ProVerif shows the same result obtained for Chaum protocol. Namely, it shows that *Weak Anonymity*, and *Strong Anonymity* are satisfied, and it outputs the same attack presented in Section 4.5.1 against *Unforgeability*. Again *Double Spending Identification* and *Exculpability* are not relevant. Note that, obtaining the same result for the two protocols, even that they have different payment and deposit phases, confirms that the blind signature used during the withdrawal phase plays the key role in preserving the privacy of the client, as claimed by David Chaum when he introduced it.

4.5.3 Chaum *et al.* Protocol

An offline variant of the Chaum protocol is proposed in [CFN88], called Chaum *et al.* protocol. The Chaum *et al.* protocol removes the requirement that the seller must contact the bank during every payment. This introduces the risk of double spending a coin by a client.

Description. The three phases of Chaum *et al.* protocol are described as follows.

Withdrawal Phase. To obtain an electronic coin, the client randomly chooses a, c and d, and calculates the pair $H = (\mathtt{h}(a, c), \mathtt{h}(a \oplus u, d))$, where u is the client identity, $\mathtt{h}$ is a hash function, and $\oplus$ is XOR operator. The client then proceed as in the Chaum protocol but with x (the potential coin) replaced with the pair H. Namely, the client blinds the pair H and sends it to the bank. Then the bank signs and returns it to the client. The main difference from the Chaum online protocol is that the coin has to be of the following form

$$(\mathtt{h}(a, c), \mathtt{h}(a \oplus u, d))$$

where the client identity is masked inside it. This allows the bank to reveal the identity of a double spender. In order for the bank to be sure that the client provides a message of the appropriate form, Chaum *et al.* used in [CFN88] the well known "*cut-and-choose*" technique. Precisely, the client computes n such a pair H where n is the system security parameter. The bank then selects half of them and asks the client to reveal their corresponding parameters (a, b, c and r). If n is large enough the client can cheats with a low probability.

At the end of this phase the client holds the electronic coin composed of the pair H, and the bank's signature $S = \mathtt{sign}(H, sk_B)$. The client also has to keep the random values a, c, d which are used later to spend the coin.

Payment Phase. The payment is made offline according to the following steps:

1. To make the payment, the client presents the pair H and the bank's signature S to the seller. The seller checks the signature, if it is correct then he chooses and sends a random binary bit y, a challenge, to the client. The client returns to the seller:
   - The values a and c if y is 0.
   - The values $a \oplus u$ and d if y is 1.
2. The seller checks the compliance of the values sent with the pair H. If everything (the signature and the values) is correct, the payment is accepted.

At the end of the payment phase, the seller holds the pair H, the signature S, the values of either (a, c) or $(a \oplus u, d)$, and the challenge y. All these data together compose the transaction the seller has to present to the bank at deposit.

Note that, in case where n pairs are used for the coin, the challenge y will be a n bits string and for each bit either the corresponding values of (a, c) or $(a \oplus u, d)$ are revealed to the seller.

Deposit Phase. To deposit a coin:

1. The seller provides to the bank the transaction (H, S, y, (a, c)) or (H, S, y, $(a \oplus u, d)$).
2. The bank checks the signature and also whether the values (a, c) or $(a \oplus u, d)$ correspond to their hash value in H. If any of these values is incorrect, the fault is on the seller's part, as he was able to independently check the regularity of the coin at payment. If the coin is correct, the bank checks its database to see whether the same coin had been used before. If it has not, the bank credits the seller's account with the appropriate amount. Otherwise, the bank rejects the transaction.

Note that, Chaum *et al.* protocol does not prevent double spending, however it preserve client's anonymity only if he spends a coin once. However, a double spender can be identified when the coin has the form $(\mathtt{h}(a, c), \mathtt{h}(a \oplus u, d))$. However, the bank can simulate the coin withdrawal and payment (as the bank knows the identities of all the clients), thus the bank can blame a honest client for double spending. As a countermeasure, the authors propose to concatenate two values z and z' with u inside the pair H to have $(\mathtt{h}(a, c), \mathtt{h}(a \oplus (u, z, z'), d))$ and provide to the bank, at withdrawal, additionally the client's signature on $\mathtt{h}(z, z')$.

Modeling in ProVerif. To model the Chaum *et al.* protocol in ProVerif, in addition to the equational theory used for the Chaum protocol (Table 4.1), we use the function `xor` to represent the exclusive or ($\oplus$) of two values. Given the first value, the second value can be obtained using the function `unxor`. Such an – admittedly limited – modeling for $\oplus$ operator is sufficient to catch the functional properties of the scheme required by Chaum *et al.* protocol, but does not catch all algebraic properties of this operator. However, there are currently no tools that support observational equivalence – which we need for the anonymity properties – and all algebraic properties of $\oplus$. Küsters *et al.* [KT11] proposed a way to extend ProVerif with $\oplus$. Their tool translates a model of the protocol to a ProVerif input where all $\oplus$ are ground terms to enable automated reasoning. However, this tool can only deal with secrecy and authentication properties, and does not support equivalence properties. The `xor` function is only used to hide the client's identity u using a random value a (*i.e.*, $a \oplus u$), which we model as $\mathtt{xor}(a, u)$. The bank then uses `a` to reveal the client's identity `u` if he double spends a coin. This is modeled by the following two equations

$$\mathtt{unxor}(\mathtt{xor}(a, u), a) = u$$
$$\mathtt{unxor}(a, \mathtt{xor}(a, u)) = u$$

which represents the various ways: $((a \oplus u) \oplus a) = u$, or $(a \oplus (a \oplus u)) = u$. We always assume that identity is the second value, and this is how we model it inside honest processes.

Analysis. As expected ProVerif confirms that *Unforgeability* is not satisfied, a corrupted client can double spend a coin. In fact the seller cannot know whether a certain coin is already spent or not, he accepts any coin that is certified by the bank. However, a collusion between a client and the attacker cannot lead to forging a coin.

In case of double spending, the bank may receive two transactions of the form $tr_1 = (h, hx, sign((h, hx), skB), 0, a, c)$ and $tr_1 = (h, hx, sign((h, hx), skB), 1, \mathtt{xor}(a, u), d)$. The bank can apply a test to obtain the identity u. This is done using the `unxor` function as $\mathtt{unxor}(\mathtt{xor}(a, u), a) = u$. The evidence here is showing that the identity of the client is masked inside the coin. This can be done thanks to the values of $(a, c, \mathtt{xor}(a, u), d)$ which are initially known only to the client. Spending the coin only once reveals either (a, c) or $(\mathtt{xor}(a, u), d)$ which does not allow to obtain the identity u. Note that, if the two sellers provide the same challenge the two transactions are exactly equal. In this case no double spending is detected and the second transaction will be rejected by the bank. Actually, the bank cannot know whether a client double spends the same coin (with same challenge provided in the two spends), or the seller deposits again another copy of the same transaction. In practice this can be avoided with high probability if n pairs coin ("cut-and-choose" technique) is used and thus n bits challenge. However, considering only one pair there is a probability of one to half (at most) for the bank to identify a double spender (cases where same challenge is provided in both spends are eliminated).

In ProVerif, we model the output of an identity and an evidence of the test $\mathtt{T}_{\mathtt{DSI}}$ by an emission of the *event OK*, and *event KO* otherwise. To say that *Double Spending Identification* is satisfied we should have that the test $\mathtt{T}_{\mathtt{DSI}}$ does not emit the *event KO* for every two valid transactions tr_1, tr_2 that are different but involves the same coin, *i.e.*, it always emits *event OK* for such transactions. Note that, our test $\mathtt{T}_{\mathtt{DSI}}$ does not capture the case explained above where the two transactions have the same challenge since same coin and same challenge results in same transaction. Regardless of that, ProVerif shows that the test can emit the *event KO* for certain two transactions satisfying the required conditions. Actually, a corrupted client can withdraw a coin that does not have the appropriate form (*e.g.*, client's identity is not masked inside it), thus the bank cannot obtain the identity in case of double spending. Note that, if the bank only certifies coins with the appropriate form at withdrawal (*i.e.*, of the form $(\mathtt{h}(a, c), \mathtt{h}(a \oplus u, d))$), then the property holds, ProVerif confirms that. Again, in practice applying the "cut-and-choose" technique can guarantee with high probability that the coin is in the appropriate form. However, applying this technique using Proverif does not make any difference since ProVerif works under symbolic model which deals with possibilities and not with

Property	Result	Time
Unforgeability	×	<1s
Double Spending Identif.	×	<2s
*Double Spending Identif.**	✓	<2s
*Exculpability**	×	< 6s
Exculpability†	✓	< 6s
Weak Anonymity	✓	<1s
Strong Anonymity	✓	<1s

TABLE 4.3: Results of our analysis on the formal model of the Chaum *et al.* protocol. (✓) indicates that the property holds. (×) indicates that it fails (ProVerif shows an attack). (∗) Only coins with the appropriate form are considered. (†) After applying the countermeasure.

probabilities. For instance, the attacker still can guess the pairs that the bank requests to reveal and construct them in the appropriate form but cheats with the others which composes the coin.

We analyze *Exculpability* in the case where only coins of appropriate form are considered *i.e.*, the case where *Double Spending Identification* holds. ProVerif confirms that a corrupted bank can blame a honest client. The bank can simulate the withdrawal and the payment since the bank knows the identity of the client. Thus obtaining two transactions satisfying the required conditions. This is due to the fact that the evidence obtained by the test, which is showing that the client's identity is masked inside the coin, is not strong enough to act as a proof. However, the attacker cannot re-spend a coin withdrawn and spent by a honest client. After applying the countermeasure that is including some terms z and z' so that the client signs $\mathtt{h}(z, z')$. ProVerif confirms that *Exculpability* holds. Applying the countermeasure results in a new test which takes, in addition to the two transactions, the client's signature on $\mathtt{h}(z, z')$. The test shows, in case of double spending, that the identity u and the preimage (z, z') of the hash signed by the client are masked inside the coin. This represents a stronger evidence which acts as a proof that the client withdrew the coin since the bank cannot forge the client's signature (under perfect cryptographic assumption).

We note that, Ogiela *et al.* [OS14] show an attack on Chaum *et al.* protocol: when a client double spends a coin, the sellers can forge additional transactions involve the same coin, so that the bank cannot know how many transactions are actually result from spends made by the client and how many are forged by the sellers. In such a case according to our definition *Unforgeability* does not hold since the client have at least to spends the coin twice. But, corrupted sellers can blame a corrupted client who double spends a coin for further spends. Moreover the bank can still identify the client and punish him as the bank can be sure that he at least spent the coin twice.

Concerning privacy properties, ProVerif shows that Chaum *et al.* protocol still satisfies

both *Weak Anonymity* and *Strong Anonymity*. Note that (for honest client), we fix the value of the challenge. Otherwise, since ProVerif internally repeats actions, the attacker can use all possible challenges, and thus leads to double spending a coin by the (honest) client which reveals his identity at payment.

To sum up, ProVerif confirms the claim about preserving client's anonymity. ProVerif also was able to show that a client can double spend a withdrawn coin but cannot forge a coin, and that the bank can identify the double spender if the coin is in the appropriate form. ProVerif also shows, in case where the coins are in the appropriate form, that the bank can simulate a withdrawal and payment, and thus can blame him for double spending. After applying the countermeasure no attack against *Exculpability* is found.

4.6 Conclusion

E-cash protocols can offer anonymous electronic payment services. Numerous protocols have been proposed in the literature, and multiple flaws were discovered. To avoid further bad surprises, formal verification can be used to improve confidence in e-cash protocols. In this chapter, we developed a formal framework to automatically verify e-cash protocols with respect to multiple essential privacy and forgery properties. Our framework relies on the Applied π-Calculus and uses ProVerif as the verification tool.

As case studies, we analyzed using ProVerif three e-cash protocols: the Chaum protocol [Cha82], the DigiCash protocol, and the Chaum *et al.* protocol [CFN88]. Our results confirm some claims and already known weaknesses. We also identified that some synchronization is necessary in case of online protocols to prevent double spending.

As future work, we would like to investigate further case studies and to extend our model to cover transferable protocols with divisible coins. Also we would like to use the tool SAPIC based on Tamarin, in order to see how it can help to analyze e-cash protocols. Furthermore, as we did not model the details of "cut-and-choose" technique of the Chaum *et al.* protocol, one could perform an analysis in the computational world in order to take this into account.

Chapter 5

e-Reputation Protocols

Reputation protocols are tools to quantify the trustworthiness of users (or entities) based on their past behavior. In such protocols, for each user a reputation score, which reflects the trust of the others on him, is computed based on the opinions of other users. Thus, for reputation protocols to serve the purpose expected from them users must provide feedbacks that reflect their opinions. Moreover, reputation scores must be correctly computed from these feedbacks. In order for users to provide honest feedbacks without a fear from a potential revenge their privacy must be preserved, and to trust reputation scores they must be verified. In this chapter, we model reputation protocols in Applied π-Calculus, and formally define several related authentication and privacy properties. Then, we define two verifiability properties in an abstract way. Finally, we validate our model by analyzing, using ProVerif, the security of a simple reputation protocol presented in [PRT04].

Contents

5.1 Introduction

A reputation protocol computes and publishes reputation scores for a set of entities (*e.g.*, service providers, goods) within a community, based on a collection of opinions that the users provide about these entities. In a typical reputation protocol, a user interacts with a certain entity, and then provides a rate that reflects his satisfaction on this entity. The reputation score of that entity is then computed from all the rates provided about it. In this sense, reputation is a quantity derived from the underlying network, and which is globally visible to all members of the network [Fre78]. An example of a reputation protocol is the one used by eBay: after a buyer buys a good from a seller, both the buyer and the seller may provide a rate (+1, 0, or -1). The reputation score of a user is simply the sum of all the rates he received. For most transactions, the buyer can additionally provide a detailed rate reflecting his satisfaction about good description, communication, shipping time and charges.

Reputation systems have been implemented in e-commerce systems, such as e-Bay, and have been contributed to the success of these systems [RKZF00]. Without reputation mechanisms, opportunistic behavior of some users can lead to peer mistrust and eventual system failure [Ake70]. For example, Resnick and Zeckhauser have analyzed the e-Bay reputation system and have concluded that the system does seem to encourage transactions [RZ02]. Moreover, several researches have found that sellers reputation has significant influences on auction prices [HW06]. An important issue with reputation systems is that the users hesitate to submit negative feedbacks when these feedbacks are public [RZ02]. A main reason behind such behavior is that the users have the fear of future revenge. Having honest feedbacks, reputation based-systems help users to decide who to trust, encourage trustworthy behavior, and detect dishonest users. In order to incentivize the users to provide honest rates, a reputation protocol has to preserve their anonymity.

Further privacy and security properties are also significant for reputation protocols as they can be affected by several adversarial behaviors. For instance, a user may provide himself very positive feedback. Assuming that a user is not eligible to give himself a feedback, property *Rate Origin Authentication*, which states that only eligible users can give rates (at most one each), fails if a user rates himself or if Moreover, property *User Eligibility Verification* allows one to verify that only eligible users gave rates. Another interesting scenario is when two users exchange positive rates. We can limit such a scenario by preserving the anonymity of the rates and ensuring recipient-freeness, which are capable of prohibiting a user from proving to the another that he rates him positively. Note that for a reputation protocol, only users that interacts with, *e.g.*, a service providers should

be eligible to rate him. This prevents users from collude, for instance, to advertise the reputation a certain user more than its real value, known as *ballot-stuffing* attack [Del00].

Note that, our properties cannot detect attacks such as bad-mothing and ballot-stuffing attacks, but properties such as: *User Eligibility Verification*, *Receipt-Freeness*, and *Coercion-Resistance* can minimize them. As *User Eligibility Verification* imposes some constraints on the users in order to provide a rate (*e.g.*, accomplish a successful interaction); *Receipt-Freeness* can limit the bribing and positive rate exchange between users as they cannot provide a proof that they provided a certain rate; similarly does *Coercion-Resistance* even when interacting with a coercer.

Contributions. Toward security analysis of reputation protocols we provide, in this chapter, the following contributions:

- We model reputation protocols in the Applied π-Calculus [AF01]. Then, we propose formal definitions of eight authentication and privacy properties of reputation protocols.
- We define two verifiability properties that allow us to verify users eligibility and reputation scores correctness.
- We discus our definitions by analyzing using ProVerif the simple protocol proposed in [PRT04].

Outline of the Chapter. We discuss the related work in Section 5.2. We formally model reputation protocol in Applied π-Calculus, and define eight authentication and privacy properties in Section 5.3. Then in Section 5.4, we give high-level definitions of two verifiability properties related to e-reputation protocols. In Section 5.5, we analysis the protocol by Pavlov *et al.* [PRT04]. Finally, we conclude in Section 5.6.

5.2 Related Work

Reputation Protocols. Marsh [Mar94] has provided one of the first formal treatment of trust. However, the proposed model emphasis on users' own experiences rather than allowing them to collectively build the trustworthiness in each other. Moreover, it is complex to be implemented in today's systems. Latter, Rahman and Hailes [AH00] have simplified and adopted the Marsh's model to be implemented in peer-to-peer networks. However, the developed approach suffers from scalability problems, and requires every user to keep a knowledge about the whole network. Aberer and Despotovic have proposed an approach that adresses reputation-based trust management in the context of peer-to-peer systems [AD01]. The proposed approach allows to assess trust by computing a user reputation from his previous interactions with other users.

In the other hand, several (claimed) secure reputation protocols have been proposed. Androulaki *et al.* [ACBM08] have proposed a reputation protocol that relies on trusted central authority to demonstrate the validity of rates. Pavlov *et al.* [PRT04] have proposed a decentralized reputation protocol to preserve users privacy. Others have tried to have privacy preserving protocols [AGLM+13a, Ste06, Ste08] by dealing with *untraceability* that is to ensure that an attacker cannot distinguish whether the same user is involved in two interactions or not. Bethencourt *et al.* [BSS10] have formally defined the anonymity of both the users and the target, and propose a protocol argued informally to satisfy those definitions. Anceaume *et al.* [AGLM+13b] have extended their work to handle non-monotonic ratings and mention additional security properties concerned in reputation scores correctness. However, to best of our knowledge no general formal framework that allows the verification of the security properties in e-reputation protocols have been given. In some related domains, there are numerous papers presenting the formalization and verification of the security properties, for instance in e-voting [DLL11, DLL12b, BHM08, DKR09], in e-auction systems [DJP10, DLL13, DJL13].

Link to Voting Systems. Some of the security properties we present herein for reputation protocols seem to relate with those studied in other applications such as voting. For instance, user eligibility is analogous to voter eligibility, and rate privacy reminds vote privacy. However, still there are fundamental differences between the two applications. A main difference between the two applications is that in voting normally each voter can cast only one vote, while in reputation each user can rates several users. This affects the adversary model. For instance, in voting a collusion between two candidates by exchanging votes provides no gain for any of them as each of them loses his vote in this process. While in reputation exchanging positive rates benefits both users. Moreover as each user can give several rates, properties such as untraceability are interesting in reputation protocols, but we do not find such a property in voting as each votes can cast at most one vote. Note also, in reputation each user can play at the same time both roles a (normal) user and a target.

Furthermore, in voting the candidates and the voters (and thus the maximum possible number of votes) are already known, and after voting process the total number of votes and that taken by each candidate will be publicly available. Thus, there is a certain leakage of information. For example, if a candidate does not receive any vote, the attacker can exclude this previously possible option. While in reputation all users can rate each others playing two rules at the same time (rate provider and target). Also, the number of provided rates is not (necessarily) publicly known. Thus, having a reputation score does not give us any information about the number of the rates provided by the users. Note that, in reputation protocols that support both negative and positive rates, a score zero does not necessarily means zero rates. Even in case of only positive rates, a score

zero might be due to the fact that this user did not make any interaction with the others yet, not necessarily means that users provide their rates to other targets like in voting. Actually, in reputation a user do not have to choose between different targets as he can provides rates for all users he interacted with. Note also that, providing a rate for a certain user is not always good like when you vote for a certain candidate in voting, as the rate could be a bad one.

5.3 Authentication and Privacy in Reputation

In this section, we model reputation protocols in the Applied π-Calculus, then we propose formal definitions for two authentication and six privacy properties.

5.3.1 Modeling Reputation Protocols in the Applied π-Calculus

A reputation protocol may involves users who interact and provide rates about each other, and authorities who often handles the rates, and calculates and publishes the reputation scores. Without loss of generality, we distinguish the user that provides the service, the *target*, from the user that provides a feedback on the target or the service, the *normal user* (in short the *user*). A *reputation protocol* specifies the processes executed by users, targets, and authorities.

Definition 5.1. (Reputation Protocol in the Applied π-Calculus). *A* reputation protocol *is defined by a tuple* $(U, T, A_1, \ldots, A_l, \tilde{n}_p)$*, where* U *is the process executed by the users,* T *is the process executed by the targets,* A_i*'s are the processes executed by the authorities, and* $\tilde{n}_p$ *is the set of the private channels used in the protocol.*

The process U is instantiated for each user with different, *e.g.*, keys, identity, and rates. Similarly, the process T is instantiated for each target. Note that, if the target is an object (*e.g.*, service, or good) or does nothing then T is simply the null process. A protocol can have several authorities, for example a registrar, a collector, a publisher or has no authority at all. In the latter case, again we have that A_i is equal to null process. Note also that, even decentralized protocols may require an agent that initiates a reputation score calculation for a certain target. Such an agent is considered as an authority. To reason about privacy properties, we define a reputation process as an instance of a reputation protocol.

Definition 5.2. (Reputation Process). *Let $(U, T, A_1, \ldots, A_l, \tilde{n}_p)$ be a reputation protocol, then we define a* reputation process *as the following closed plain process:*

$$\begin{aligned}\nu\tilde{n}.(A_1 \mid \ldots \mid A_{l'} \mid T\sigma_{idt_1} \mid \ldots \mid T\sigma_{idt_m} \mid \\ (U\sigma_{idu_1}\sigma_{r_{11}}\sigma_{idt_{11}} \mid \ldots \mid U\sigma_{idu_1}\sigma_{r_{1p_1}}\sigma_{idt_{1p_1}}) \mid \\ \vdots \\ \mid (U\sigma_{idu_k}\sigma_{r_{k1}}\sigma_{idt_{k1}} \mid \ldots \mid U\sigma_{idu_k}\sigma_{r_{kp_k}}\sigma_{idt_{kp_k}}))\end{aligned}$$

where $\tilde{n}$ is the set of all restricted names, which includes some of the protocol's private channels $\tilde{n}_p$; $A_1, \ldots, A_{l'}$ $(l' \leqslant l)$ are the processes executed by the honest authorities; $T\sigma_{idt_i}$ is the process executed by the target idt_i; $U\sigma_{idu_i}\sigma_{r_{ij}}\sigma_{idt_{ij}}$ is the process executed by the user idu_i, which rates the target idt_{ij} with the rate r_{ij}. Note that, idu_i can provide p_i rates on different targets.

A party can play both roles of user and target in a reputation process as both processes U and T can be instantiated with the same identity. Note that, in some protocols a certain property may hold even if some of the authorities are dishonest, while other protocols may require all the authorities to be honest. During the analysis, we choose which authorities $A_1, \ldots, A_{l'}$ are honest by including them in the reputation process. Only honest parties are modeled in a reputation process. Dishonest parties are left to the attacker. The attacker has complete control to the network, except private channels (*e.g.*, Dolev-Yao attacker [DY83a]).

To capture threats due to bribing and coercion, we consider corrupted parties P^{ch_1} and P^{ch_1,ch_2} from [DKR09], which we respectively recalled by the Definitions 2.1 and 2.2 in Section 2.1.1. The process P^{ch_1} is a variant of P which shares with the attacker channel ch_1. Through ch_1, P^{ch_1} sends all its inputs and freshly generated names (but not restricted channel names). The process P^{ch_1,ch_2} does not only reveal the secret data on channel ch_1, but also takes orders from the attacker on the channel ch_2 before sending a message or branching. Given a reputation process any process P can be replaced by either P^{ch_1} or P^{ch_1,ch_2}. Note that, in particular we use P^{ch_1} and P^{ch_1,ch_2} to respectively define *Receipt-Freeness* and *Coercion-Resistance*. For the sake of simplicity in notations, we define a *reputation context* which we use inside our the definitions of our properties.

Definition 5.3. (Reputation Context). *Given a reputation process, a subset of users I_U, and a subset of targets I_T, we define the* reputation context *$RP_I[\_]$, where $I = I_U \cup I_T$, as follows:*

$$RP_I[\_] \equiv \nu\tilde{n}.(A_1 \mid \ldots \mid \underset{idt_i \notin I_2}{A_{l'} \mid T\sigma_{idt_i}} \mid \underset{idu_i \notin I_1}{\_} \mid (U\sigma_{idu_i}\sigma_{r_{i1}}\sigma_{idt_{i1}} \mid \ldots \mid U\sigma_{idu_i}\sigma_{r_{ip_i}}\sigma_{idt_{ip_i}}))$$

A reputation context $RP_I[\_]$ is the process RP without all the processes of the parties included in the set I; they are replaced by "holes". We use this notation, for instance, to enumerate all the sessions executed by the user idu_1 without repeating the entire reputation instance; in this case, we can rewrite the reputation instance as $RP_{\{idu_1\}}[U\sigma_{idu_1}\sigma_{r_{11}}\sigma_{idt_{11}} \mid \ldots U\sigma_{idu_1}\sigma_{r_{1p_1}}\sigma_{idt_{1p_1}}]$.

To define authentication properties, we use the following events:

- Event $\texttt{sent}(idu, idt, r)$ emitted when the user idu sends a rate r (or sum of rates) for the authority (or responsible party) to evaluate the target idt. This event is emitted just before sending the message containing the rate.
- Event $\texttt{record}(idu, idt, r)$ emitted when the rate r (or sum of rates) from the user idu provided about the target idt is received by the authority (or the intended party). This event is placed after receiving the rate and performs the required checks before accepting it, if any.
- Event $\texttt{eligible}(idu, idt)$ emitted when the user idu is certified as an eligible user to provide a rate about the target idt. It is placed just before providing the credential by the responsible party.

5.3.2 Authentication Properties

We define two authentication properties *Rate Integrity* and *Rate Origin Authentication*. After placing the events inside the reputation processes, the authentication properties can then be defined as follows.

Rate Integrity ensures that the rate is not altered and received by the responsible party as provided by the user.

Definition 5.4. (Rate Integrity). *A reputation protocol ensures* Rate Integrity *if for any reputation process RP on every possible execution trace, each occurrence of the event* $\texttt{record}(idu, idt, r)$ *is preceded by a distinct occurrence of the corresponding event* $\texttt{sent}(idu, idt, r)$.

Rate Origin Authentication ensures that only the users that have a certain credential can provide a rate. The credential can be a certificate from an authority, an entry in a database, or a proof which indicates that he has been interacted with the target.

Definition 5.5. (Rate Origin Authentication). *A reputation protocol ensures* Rate Origin Authentication*, if for any reputation process RP on every possible execution trace, each occurrence of the event* $\texttt{record}(idu, idt, r)$ *is preceded by a distinct occurrence of the corresponding event* $\texttt{eligible}(idu, idt)$.

5.3.3 Privacy Properties

We define six privacy properties: *User Privacy*, *Rate Privacy*, *Rate Anonymity*, *Untraceability*, *Receipt-Freeness*, and *Coercion-Resistance*.

Definition 5.6. (User Privacy). *A reputation protocol ensures* User Privacy *if for any reputation process RP, any two users idu_1, idu_2, any target idt, and any rate r, we have that:* $RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_r\sigma_{idt}|T\sigma_{idt}] \approx_l RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_2}\sigma_r\sigma_{idt}|T\sigma_{idt}]$.

The intuition is that if an attacker cannot detect if a rate is provided by a honest user idu_1 or by idu_2, then he cannot know anything about the users idu_1 and idu_2. Note that this definition is robust even in situations where only two users (idu_1 or idu_2) provide rates, *i.e.*, the context $RP_{\{idu_1,idu_2,idt\}}[\_]$ contains no honest user.

As already noted, in some protocols the *User Privacy* may hold even if the authorities are dishonest, while other protocols may require the authorities to be honest. When proving privacy, we choose which authorities we want to model as honest, by including them inside the context $RP_{\{idu_1,idu_2,idt\}}[\_]$. It is also particularly interesting for *User Privacy* to consider a corrupted target t. This results in a similar definition but with a process $(T\sigma_{idt})^{ch_1,ch_2}$ instead of $T\sigma_{idt}$. We can also consider corrupted users other than idu_1 and idu_2, as well as, other corrupted targets than t.

Note that, *User Privacy* is a strong property to be satisfied by a protocol since user's identities may be revealed for an authority, and the target may knows the identity of all the users interacted with him, and thus a set includes all those that rate him. A similar property is *Rate Privacy*, which says that an attacker cannot know the rate provided by a certain user.

Definition 5.7. (Rate Privacy). *A reputation protocol ensures* Rate Privacy *if for any reputation process RP, any user idu, any target idt, and any two rates r, r', we have that:* $RP_{\{idu,idt\}}[U\sigma_{idu}\sigma_r\sigma_{idt}|T\sigma_{idt}] \approx_l RP_{\{idu,idt\}}[U\sigma_{idu}\sigma_{r'}\sigma_{idt}|T\sigma_{idt}]$.

Rate Privacy states that two processes with different rates have to be observationally equivalent. Again, we can consider corrupted targets and corrupted users other than *idu*. Note that like *User Privacy*, *Rate Privacy* is also strong. Consider for instance the case where there is only one rate, then as the reputation score is published the rate provided by the honest user can be obtained. A weaker property is *Rate Anonymity* which says that an attacker cannot associate a certain rate to its corresponding user. Note that, an attacker might know all users' identities and all rates, but he cannot link between a user and his rate.

Definition 5.8. (Rate Anonymity). *A reputation protocol ensures* Rate Anonymity*, if for any reputation process RP, any users idu_1, idu_2, any target idt, and any rates r_1, r_2, we have that:*

$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$$
$$\approx_l$$
$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_2}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$$

Rate Anonymity states that the process where user idu_1 provides a rate r_1 and user idu_2 provides a rate r_2 is equivalent to the process where idu_1 provides a rate r_2 and idu_2 provides a rate r_1. This prevents the attacker from obtaining the identity of the user who provides a certain rate. Note that, a protocol that ensures *User Privacy* also ensures *Rate Anonymity*. Similarly, a protocol that ensures *Rate Privacy* also ensures *Rate Anonymity*.

Theorem 5.1. *A reputation protocol that satisfies* User Privacy*, also satisfies* Rate Anonymity*.*

Proof. Suppose that a reputation process satisfies *User Privacy*, then we have that

$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_r\sigma_{idt}|T\sigma_{idt}] \approx_l RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_2}\sigma_r\sigma_{idt}|T\sigma_{idt}]$$

Let $RP'_{\{idu_1,idu_2,idt\}}[\_] \equiv RP_{\{idu_1,idu_2,idt\}}[\_|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$ be an reputation context where $RP_{\{idu_1,idu_2,idt\}}[\_]$ is any reputation context, idu_1 and idu_2 are any two users, r is any rate, and t is any target. Then, by *User Privacy*, we have that:

$$RP'_{\{idu_1,idu_2,idt\}}[U\sigma_{\mathbf{idu_1}}\sigma_{r_1}\sigma_{idt}] \approx_l RP'_{\{idu_1,idu_2,idt\}}[U\sigma_{\mathbf{idu_2}}\sigma_{r_1}\sigma_{idt}]$$

Thus, by substituting $RP'_{\{idu_1,idu_2,idt\}}[\_]$ with its value we get:

$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{\mathbf{idu_1}}\sigma_{r_1}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$$
$$\approx_l$$
$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{\mathbf{idu_2}}\sigma_{r_1}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$$

Similarly, we have that

$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|U\sigma_{\mathbf{idu_2}}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$$
$$\approx_l$$
$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|U\sigma_{\mathbf{idu_1}}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$$

Hence, we can deduce that

$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$$
$$\approx_l$$
$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|U\sigma_{idu_1}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$$

Therefore, *Rate Anonymity* is satisfied by the reputation process. □

Theorem 5.2. *A reputation protocol that satisfies* Rate Privacy*, also satisfies* Rate Anonymity*.*

Proof. Suppose that a reputation process satisfies *Rate Privacy*, then we have that

$$RP_{\{idu,idt\}}[U\sigma_{idu}\sigma_r\sigma_{idt}|T\sigma_{idt}] \approx_l RP_{\{idu,idt\}}[U\sigma_{idu}\sigma_{r'}\sigma_{idt}|T\sigma_{idt}]$$

Let $RP'_{\{idu_1,idu_2,idt\}}[\_] \equiv RP_{\{idu_1,idu_2,idt\}}[\_|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}]$ be an reputation context where $RP_{\{idu_1,idu_2,idt\}}[\_]$ is any reputation context, idu_1 and idu_2 are any two users, r is any rate, and t is any target. Then, by *Rate Privacy*, we have that:

$$RP'_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{\mathbf{r_1}}\sigma_{idt}] \approx_l RP'_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{\mathbf{r_2}}\sigma_{idt}]$$

Thus, by substituting $RP'_{\{idu_1,idu_2,idt\}}[\_]$ with its value we get:

$$\begin{gathered} RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{\mathbf{r_1}}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}] \\ \approx_l \\ RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{\mathbf{r_2}}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}] \end{gathered}$$

Note that, only the rate provided by the user idu_1 is changed from r_1 in the left side to r_2 in the right one. The processes of idu_2 and t are the same on both sides. Similarly, we have that

$$\begin{gathered} RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_2}\sigma_{idt}|U\sigma_{idu_2}\sigma_{\mathbf{r_2}}\sigma_{idt}|T\sigma_{idt}] \\ \approx_l \\ RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_2}\sigma_{idt}|U\sigma_{idu_2}\sigma_{\mathbf{r_1}}\sigma_{idt}|T\sigma_{idt}] \end{gathered}$$

Hence, we can deduce that

$$\begin{gathered} RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_2}\sigma_{idt}|T\sigma_{idt}] \\ \approx_l \\ RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_2}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}] \end{gathered}$$

Therefore, *Rate Anonymity* is satisfied by the reputation process. □

The next property is *Untraceability* which states that an attacker cannot decide whether or not two rates have been produced by the same user.

Definition 5.9. (Untraceability). *A reputation protocol ensures* Untraceability *if for any reputation process RP, any two users* idu_1*,* idu_2*, any targets* idt*,* $idt_{11}, \ldots, idt_{1m_1}$*,*

$idt_{21}, \ldots, idt_{2m_2}$, and any rates r, $r_{11}, \ldots, r_{1m_1}$, $r_{21}, \ldots, r_{2m_2}$, we have that:

$$RP_{\{idu_1,idu_2,idt\}}[\mid_{0\leq i\leq m_1} U\sigma_{idu_1}\sigma_{r_{1i}}\sigma_{idt_{1i}} \mid_{0\leq i\leq m_1} T\sigma_{idt_{1i}} \mid_{0\leq i\leq m_2} U\sigma_{idu_2}\sigma_{r_{2i}}\sigma_{idt_{2i}} \mid_{0\leq i\leq m_2} T\sigma_{idt_{2i}} \mid U\sigma_{idu_1}\sigma_r\sigma_{idt}]$$

$$\approx_l$$

$$RP_{\{idu_1,idu_2,idt\}}[\mid_{0\leq i\leq m_1} U\sigma_{idu_1}\sigma_{r_{1i}}\sigma_{idt_{1i}} \mid_{0\leq i\leq m_1} T\sigma_{idt_{1i}} \mid_{0\leq i\leq m_2} U\sigma_{idu_2}\sigma_{r_{2i}}\sigma_{idt_{2i}} \mid_{0\leq i\leq m_2} T\sigma_{idt_{2i}} \mid U\sigma_{idu_2}\sigma_r\sigma_{idt}]$$

Untraceability ensures that the process in which the user idu_1 provides $m_1 + 1$ rates, while idu_2 provides m_2 rates, is equivalent to the process in which the user idu_1 provides m_1 rates, while idu_2 provides $m_2 + 1$ rates. The intuition is that an attacker cannot distinguish whether idu_1 or idu_2 provides a rate r even if both users previously provided some rates. Note that, a protocol that satisfies *Untraceability* also satisfies *User Privacy*, as *User Privacy* is a special case of *Untraceability* for $m_1 = m_2 = 0$, *i.e.*, when the two honest users do not provide any previous rates.

For *Receipt-Freeness* and *Coercion-Resistance* we follow the definitions introduced in [DKR09]. A protocol is receipt-free, if an attacker cannot distinguish between a situation where a user idu_1 provides a rate r_c according to the attacker's wishes and reveals his data on a channel ch, and a situation where idu_1 actually provides a rate r_1 of his choice and pretends to reveal his secret data (this is modeled by process U'). The process U' is a process in which user idu_1 provides a rate r_1, but communicates with the attacker (coercer) to trick him by saying that his desired rate r_c is provided. This can be done by providing the attacker a fake receipt, *e.g.*, using a trapdoor to generate a different opening key.

Definition 5.10. (Receipt-Freeness). *A reputation protocol ensures* Receipt-Freeness *if for any reputation process RP, any users idu_1, idu_2, any target idt, and any rates r_1, r_c, there exists a closed plain process U' such that:*

- $U'^{\backslash out(ch,.)} \approx_l U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}$, *and*
- $RP_{\{idu_1,idu_2,idt\}}[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$
  $\approx_l RP_{\{idu_1,idu_2,idt\}}[U'|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]$.

The user idu_2 is needed to provide a counterbalancing rate. Note that, *Receipt-Freeness* is stronger than *Rate Anonymity*.

Theorem 5.3. *A reputation protocol that satisfies* Receipt-Freeness*, also satisfies* Rate Anonymity.

Proof. Suppose that a reputation process satisfies *Receipt-Freeness* then, by Definition 5.10, there exists a closed plain process U' such that:

- $U'^{\backslash out(ch,.)} \approx_l U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}$, and (1)

– $RP_{\{idu_1,idu_2,idt\}}[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$
$\approx_l RP_{\{idu_1,idu_2,idt\}}[U'|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]. \quad (2)$

Thus, by applying the evaluation context $\mathcal{C} \equiv \nu ch.(\_|\ !in(ch,x))$ on both sides of (2), we get[1]:

$$RP_{\{idu_1,idu_2,idt\}}[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]^{\backslash out(ch,\cdot)}$$
$$\approx_l$$
$$RP_{\{idu_1,idu_2,idt\}}[U'|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]^{\backslash out(ch,\cdot)}$$

In the other hand, we have, by Lemma 2.2, that

$$\mathcal{C}[RP_{\{idu_1,idu_2,idt\}}[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]]$$
$$\equiv$$
$$RP_{\{idu_1,idu_2,idt\}}[\mathcal{C}[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]]$$

Thus,

$$RP_{\{idu_1,idu_2,idt\}}[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]^{\backslash out(ch,\cdot)}$$
$$\equiv$$
$$RP_{\{idu_1,idu_2,idt\}}[((U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch})^{\backslash out(ch,\cdot)}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$$

Similarly, we get:

$$RP_{\{idu_1,idu_2,idt\}}[U'|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]^{\backslash out(ch,\cdot)}$$
$$\equiv$$
$$RP_{\{idu_1,idu_2,idt\}}[(U')^{\backslash out(ch,\cdot)}|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]$$

Hence, by comparison on ($\equiv$), we have that:

$$RP_{\{idu_1,idu_2,idt\}}[((U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch})^{\backslash out(ch,\cdot)}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$$
$$\equiv$$
$$RP_{\{idu_1,idu_2,idt\}}[(U')^{\backslash out(ch,\cdot)}|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]$$

Lastly, since $U'^{\backslash out(ch,.)} \approx_l U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}$ by (1), and $(P^{ch})^{\backslash out(ch,\cdot)} \approx_l P$ for any closed plain process P such that $ch \notin fn(P) \cup bn(P)$ by Lemma 2.1, we get:

$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$$
$$\equiv$$
$$RP_{\{idu_1,idu_2,idt\}}[U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]$$

Therefore, *Rate Anonymity* is satisfied by the protocol. □

[1] Note that, we have $A^{\backslash out(ch,\cdot)} \equiv \nu ch.(A \mid !in(ch,x))$ by Definition 2.3

A property stronger than *Receipt-Freeness* is *Coercion-Resistance* which preserves rate anonymity in the case where the attacker cannot only ask for a receipt, but is also allowed to interact with the user during the rating process and to provide the messages the user should send.

Definition 5.11. (Coercion-Resistance). *A reputation protocol ensures* Coercion-Resistance *if for any reputation process RP, and any rates r_1, r_c there exists a closed plain process U' such that for any σ_r, a plain process P, and context $\mathcal{C} = \nu ch_1.\nu ch_2.(\_|P)$ satisfying $\tilde{n} \cap fn(\mathcal{C}) = \emptyset$ and*

$$RP_I[\mathcal{C}[(U\sigma_{idu_1}\sigma_r\sigma_{idt})^{ch_1,ch_2}]|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$$
$$\approx_l$$
$$RP_I[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$$

where $I = \{idu_1, idu_2, idt\}$, we have that:

- $\mathcal{C}[U']^{\setminus out(ch,.)} \approx_l U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}$, *and*
- $RP_I[\mathcal{C}[(U\sigma_{idu_1}\sigma_r\sigma_{idt})^{ch_1,ch_2}]|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}] \approx_l RP_I[\mathcal{C}[U']|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]$.

The context $\mathcal{C}$ models the attacker's behaviors which tries to force the user to provide a rate r_c. Note that, no matter what rate the user idu_1 intends to provide (σ_r), the attacker will force him to provide the rate r_c. U' is a process that provides a rate r_1 of user's choice and at the same time fakes the attacker. *Coercion-Resistance* is satisfied when the attacker cannot distinguishes between process U' and a really coerced voter. Again a counterbalancing rate is required.

Theorem 5.4. *A reputation protocol that satisfies* Coercion-Resistance*, also satisfies* Receipt-Freeness.

Proof. Suppose that a reputation process satisfies *Coercion-Resistance*, and let $\mathcal{C} = \nu ch_1.\nu ch_2.(\_|P)$ be an evaluation context from some plain process P such that

$$RP_I[\mathcal{C}[(U\sigma_{idu_1}\sigma_r\sigma_{idt})^{ch_1,ch_2}]|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}]$$
$$\approx_l$$
$$RP_I[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}] \qquad (1)$$

where $I = \{idu_1, idu_2, idt\}$. Then, by Definition 5.11, we have that:

- $\mathcal{C}[U']^{\setminus out(ch,.)} \approx_l U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}$, and
- $RP_I[\mathcal{C}[(U\sigma_{idu_1}\sigma_r\sigma_{idt})^{ch_1,ch_2}]|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}] \approx_l RP_I[\mathcal{C}[U']|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]$

Thus, by (1) and transitivity on ($\approx_l$) we get:

- $\mathcal{C}[U']^{\setminus out(ch,.)} \approx_l U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}$, and

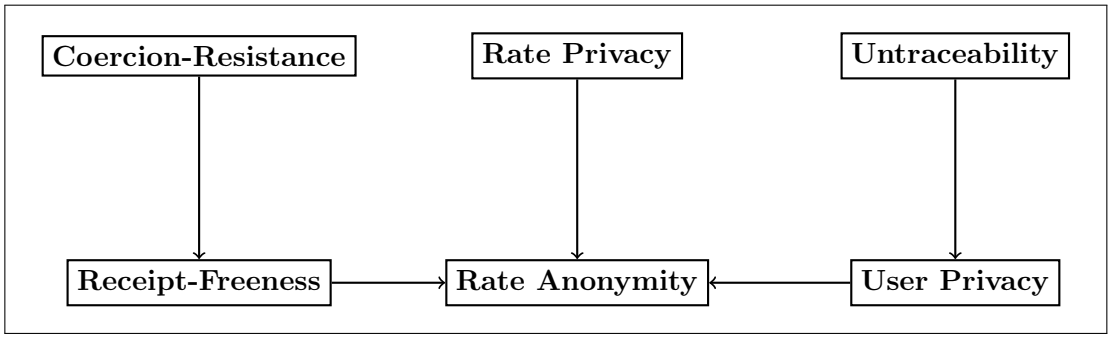


FIGURE 5.1: Relations between privacy properties of reputation protocols.

– $RP_I[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}] \approx_l RP_I[\mathcal{C}[U']|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]$

Now, let $U'' = \mathcal{C}[U']$, hence:

– $(U'')^{\backslash out(ch,.)} \approx_l U\sigma_{idu_1}\sigma_{r_1}\sigma_{idt}$, and

– $RP_I[(U\sigma_{idu_1}\sigma_{r_c}\sigma_{idt})^{ch}|U\sigma_{idu_2}\sigma_{r_1}\sigma_{idt}|T\sigma_{idt}] \approx_l RP_I[U''|U\sigma_{idu_2}\sigma_{r_c}\sigma_{idt}|T\sigma_{idt}]$

Therefore, *Receipt-Freeness* is satisfied by the protocol. □

Figure 5.1 shows all the relations we have shown between our privacy properties for reputation protocols.

5.4 Verifiability in Reputation

In this section, first we define an abstract model of reputation protocols, which defines the data involved in a reputation instance (*e.g.*, list of provided rates), as well as, some black box functions to perform some computations or checks on these data. Then, we define at high level two verifiability properties which allow a party to check the eligibility of users that provide rates and the correctness of reputation scores.

Definition 5.12. (Reputation Protocol). *A reputation protocol is a tuple* $(\mathtt{U}, \mathtt{T}, \mathtt{A}, \mathtt{L}, \mathtt{isEligible}, \mathtt{compRep}, rs)$ *where*

– $\mathtt{U}$ *is the set of users.*

– $\mathtt{T}$ *is the target.*

– A *is the authority.*

– $\mathtt{L} : \mathtt{List}(\mathtt{Rate})$ *is a list of rates provided by the users in* $\mathtt{U}$ *about the target* $\mathtt{T}$*;*

– $\mathtt{isEligible} : \mathtt{Rate} \mapsto \{true, false\}$ *is a function which takes a rate and outputs true if the rate was provided by an eligible user, and* $false$ *otherwise.*

- `compRep` *is a function that computes the reputation score given a list of rates* `L`.
- rs *is a variable referring to the reputation score assigned to the target, i.e., the published score;*

The function `compRep` may simply compute the summation of all the rates, but it may also perform more complex operations to determine the value of the reputation score. An eligible user, *e.g.*, is a user that has successfully finished an interaction with the target `T`. Note that, our definition of reputation protocol does not specify how the users and the targets interacts before the rating procedure starts. For example, the user may purchase some goods from the target.

We define the verification properties using tests. A test is an algorithm that takes as input the data visible to a participate of the reputation protocol and returns a boolean value ($true$ or $false$). The first verifiability property is *User Eligibility Verification* which allows any party to check if every counted was provided by an eligible user.

Definition 5.13. (User Eligibility Verification). *A reputation protocol ensures* User Eligibility Verification *if there exists a test* `UEV` *respecting the following conditions:*

- *Soundness:* $\texttt{UEV} = true \Rightarrow \forall r \in \texttt{L}, \texttt{isEligible}(r) = true$;
- *Completeness: if all participants follow the protocol correctly, then* $\forall r \in \texttt{L}, \texttt{isEligible}(r) = true \Rightarrow \texttt{UEV} = true$.

The second verifiability property is *Reputation Score Verification* which allows any party to verify the correctness of the target's reputation score.

Definition 5.14. (Reputation Score Verification). *A reputation protocol ensures* Reputation Score Verification *if there exists a test* `RSV` *respecting the following conditions:*

- *Soundness:* $\texttt{RSV} = true \Rightarrow rs = \texttt{compRep}(L)$;
- *Completeness: if all participants follow the protocol correctly, then* $rs = \texttt{compRep}(\texttt{L}) \Rightarrow \texttt{RSV} = true$.

5.5 Case Study: Protocol by Pavlov *et al.* [PRT04]

In this section, we applied our definitions using ProVerif on (the first) reputation protocol proposed by Pavlov *et al.* in [PRT04]. This protocol was designed to provide privacy of the rates in decentralized additive reputation systems if all users follow the protocol correctly, *i.e.*, all users are honest. It also assumes that all users provide honest rating about the target (*i.e.*, rating that correctly reflects their satisfaction), as the protocol cannot prevent the users from providing an unfair rating to increase (*resp.* decrease) the reputation score of the target more (*resp.* less) than its real value.

Description. The basic idea behind the protocol is to consider the rate provided by each user to be his secret information. The rates are cumulatively added to each other, without revealing them, to obtain a sum that represents the reputation score of the target. The protocol is initiated by a querying agent A_q looking to know the reputation score of the target, and proceeded as follows.

1. Initialization Step: the querying agent A_q orders the users in a ring: $A_q \rightarrow U_1 \rightarrow , \ldots, \rightarrow U_n \rightarrow A_q$, and sends to each user U_i the identity of his successor in the ring, *i.e.*, for $i \in \{1, \ldots, n-1\}$ it sends the identity of U_{i+1} to U_i, and for U_n it sends its identity.

2. A_q chooses a random number $r_q \neq 0$ and sends it to U_1.

3. Upon reception of r_p from his predecessor in the ring, each user U_i for $i \in \{1, \ldots, n\}$ calculates $r_p + r_i$ and sends the obtained value to his successor in the ring, where r_i is the reputation rate of the user U_i about the target.

4. Upon reception of the feedback from U_n, A_q subtracts r_q from it in order to obtain the reputation score of the target represented by the sum of all rates.

To ensure secrecy and authentication, the designers of the protocol assume an authenticated secure channel between every two users.

Formal Model. Generally, it is difficult to model arithmetic operations in formal protocol provers such as ProVerif. However, we build a simple equational theory, depicted in Figure 5.2, which handles the required arithmetics to verify the Pavlov *et al.* protocol for the case where we have two users U_1 and U_2 in addition to the querying agent. Note that, we use the same model to verify authentication and privacy properties, as well as, verifiability properties.

We model the protocol in ProVerif using a well-known equational theory for symmetric encryption (functions `senc` and `sdec`), $\texttt{senc}(m, k)$ represents the symmetric encryption of the message m with the key k and `sdec` represents the decryption function; and an equational theory for arithmetic addition and subtraction (functions `sum` and `sub`). The function `sum` takes two values x and y and return their sum $\texttt{sum}(x, y)$. Having x or y we can obtain the other one from $\texttt{sum}(x, y)$ using the function `sub`. Similarly, we can obtain $\texttt{sum}(y, z)$ and $\texttt{sum}(x, z)$ from $\texttt{sum}(\texttt{sum}(x, y), z)$ having x and y respectively. Note that with our equational theory, having x or y, one cannot obtain $\texttt{sum}(z, y)$ or $\texttt{sum}(z, x)$ from $\texttt{sum}(z, \texttt{sum}(x, y))$. Solving this requires two additional equations similar to the last two equations shown in Figure 5.2. One could ask why we do not model `sum` as a commutative function, *i.e.*, $\texttt{sum}(x, y) = \texttt{sum}(y, x)$, which solves this problem and moreover allows us to represent the subtraction using only two equations instead of four. This is due to a

ProVerif limitation as this solution causes non termination. Note that in our model, we provide the value r_p received by the user process U_i as the first argument to the function $\mathtt{sum}$, and his rate r_i as the second argument. Thus, such a term $\mathtt{sum}(z, \mathtt{sum}(x, y))$ does not appear between the messages involved in our model of the protocol.

$$\begin{aligned}
\mathtt{sdec}(\mathtt{senc}(m, k), k) &= m \\
\mathtt{sub}(\mathtt{sum}(x, y), x) &= y \\
\mathtt{sub}(\mathtt{sum}(x, y), y) &= x \\
\mathtt{sub}(\mathtt{sum}(\mathtt{sum}(x, y), z), x) &= \mathtt{sum}(y, z) \\
\mathtt{sub}(\mathtt{sum}(\mathtt{sum}(x, y), z), y) &= \mathtt{sum}(x, z)
\end{aligned}$$

FIGURE 5.2: Equational theory for our model of Pavlov *et al.* protocol.

An alternative equational theory could model the sum of two values x and y as their exclusive-or (XOR), *i.e.*, $\mathtt{xor}(x, y)$ instead of $\mathtt{sum}(x, y)$. In this case the subtraction will be represented by another XOR application, *i.e.*, $\mathtt{xor}(\mathtt{xor}(x, y), y) = x$. However, there are currently no tools that support the algebraic properties of XOR operator and at the same time support observational equivalence – needed for the privacy properties. Küsters *et al.* [KT11] have shown how to reduce the derivation problem for Horn theories with XOR to the XOR-free case. Their reduction allows one to carry out the analysis of the protocols that involve XOR operator using tools, such as ProVerif. However, their result is valid only with secrecy and authentication properties, but not with equivalence properties.

We assume that all the parties (querying agent and users) are honest. To model the authenticated secure channel, we assume that all messages are exchanged encrypted with a shared symmetric key, which ensures the secrecy of the messages. Moreover, we assume that the unique identities of the sender and the receiver are included in the message. This allows the receiver to authenticate the sender, and also ensures that the message will considered only by the intended receiver as all users are honest.

Analysis. The results of our analysis is detailed below. A summery for the results is shown in Table 5.1.

User Privacy: ProVerif shows that *User Privacy* is not satisfied, simply since the set of all users' identities that provides rates on the target is public (at least known by the querying agent).

Rate Privacy: in case where we have only one user, it is clear that we do not have *Rate Privacy* as the reputation score will be equal to the rate of this user, and thus the querying agent can knows this rate. We show, using ProVerif, that the protocol ensures *Rate Privacy* in the case of two users (other than the querying agent). As we already

Property	Result	Time
User Privacy	×	< 1 s
Rate Privacy	✓	< 1 s
Rate Anonymity	✓	< 1 s
Untraceability	×	-
Receipt-Freeness	×	-
Coercion-Resistance	×	-
Rate Origin Authentication	✓	< 1 s
*Rate Integrity**	✓	< 1 s
User Eligibility Verification	×	< 1 s
Reputation Score Verification	✓	< 1 s

TABLE 5.1: Results of our analysis on the formal model of the Pavlov *et al.* protocol. $(*)$ means that without injectivity. (-) indicates that the property is not verified using ProVerif.

mentioned, to model the authenticated channel we add the identities to the messages. Note that, if the identity of the receiver is removed from the messages, the attacker can re-direct the message sent by the first user, which contains $sum(r_q, r_1)$, to the querying agent instead of the second user, and thus the rate of the first user will be enclosed to the querying agent even if he acts honestly. A similar attack will enclose the rate of the last user (herein user two) if the attacker re-direct the first message by querying agent, which contains r_q to the last user. To make the attack on the intermediate users (in case of more than two users) the attacker needs two sessions to be initiated by the same querying agent. Note that, in all these attacks the rate will only enclosed to the querying agent and not to the attacker unless that the obtained reputation score is published by the querying agent.

Rate Anonymity: ProVerif shows that the protocol ensures *Rate Anonymity* in the case of two users (other than the querying agent).

Untraceability: since *User Privacy* is not satisfied *Untraceability* is also not satisfied.

Receipt-Freeness: the protocol does not ensure Receipt-Freeness since the shared key k can be used as a receipt which allows the attacker to enclose the value of the message received by the victim user and that sent by him, then subtract them from each other to check whether the difference is equal to the rate r_c he wants, or not. Note that, the user cannot lie about the key by giving the attacker an different key k' unless he can obtain a key $k' \neq k$ such that $r_c = sub(sdec(senc(sum(r_p, r_i), k), k'), sdec(senc(r_p, k), k'))$ where r_i is the user's rate, and r_p is the sum he received from his predecessor.

Coercion-Resistance: as *Receipt-Freeness* does not hold, then *Coercion-Resistance* also does not hold since by the Theorem 5.4 if a protocol ensures *Coercion-Resistance* then it

also ensures *Receipt-Freeness*.

Rate Origin Authentication: we show with ProVerif that *Rate Origin Authentication* is ensured by the protocol. We assume that the users included in the ring are eligible as they are registered in a certain database. Thus, the event `eligible` is emitted when the querying agent chooses the user as a node in the ring. The event `record` emitted when the rate is received by his successor in the ring (actually the summation of all previous rates is received).

Rate Integrity: we check the integrity of the messages exchanged between the parties (querying agent and users). Thus events are placed in both querying agent and user processes. ProVerif shows that the integrity of the messages is preserved if the correspondence is modeled without injectivity, and terms types (sorts) are respected. Note that, the injectivity does not hold since the attacker can re-play the message several times, and thus we will have several emissions of the event `record` preceded by only one emission of event `sent`. Note also that a flaw exists if we ignore the types of the terms: for example, the first message sent by the querying agent to the first user to inform him about the identity of his successor might be received by this user as a rate r_q, which has a different type. Thus, the event `record` will be emitted but not the event `sent`.

User Eligibility Verification: users do not provide any proof (*e.g.*, certificate) which could allow one to verify their eligibility (*i.e.*, they have been interacted with the target). Actually, the protocol assumes a set of eligible users each with a certain rate. Note that, as user eligibility is ensured, according to the definition of *User Eligibility Verification* the test that always gives true is sound and complete. However, this test is dummy as it does not provide any information for the verifier to be sure about user's eligibility.

Reputation Score Verification: the users can only get the reputation score, however they have no means to check if this score is calculated correctly from the users rates. However, if the rates are published encrypted, with a certain key whose public part is known, in a Bulletin Board then we can design a test that allows the users to verify the reputation score. The test takes users' rates and the reputation score of the target, and simply checks if the summation of all rates is equal to the score. Note that, verifiability could destroys the privacy of the rates. We show using ProVerif that such a test is sound and complete.

5.6 Conclusion

We set the first research step on the formal understanding of e-reputation systems, and establishes a framework for the analysis of their security requirements. In particular, we show how to model reputation protocols in the Applied π-Calculus, and how security

properties such as privacy, authentication, and verifiability properties can be expressed. We validate our model and definitions by analyzing, using ProVerif, the security of an e-reputation protocol, the protocol by Pavlov *et al.* [PRT04]. It has been informally argued to preserve rate privacy. Our analysis shows that it satisfies five properties, namely *Rate Privacy*, *Rate Anonymity*, and *Rate Origin Authentication*, *Rate Integrity* (without injectivity), and *Reputation Score Verification* (if users can have an access to the set of rates). While it fails to satisfy the other five: *User Privacy*, *Untraceability*, *Receipt-Freeness*, *Coercion-Resistance*, and *User Eligibility Verification*.

As a future work it would be interesting to analyze more e-reputation protocols. Several e-reputation protocols are highly depends on algebraic properties such as arithmetic operations and homomorphic encryptions, *e.g.* [PRT04, HBBS13, AGLM$^+$13b, BSS10]. Developing automatic tools that can deal with these properties is still a real challenge for the community. However, researches goes some way in the direction of finding solutions for such a problem, for instance the result obtained by Küsters *et al.* [KT11] allows us to analyze secrecy and authentication properties for protocols with XOR operator using ProVerif.

Chapter 6

Analyzing Routing Protocols in Presence of Multiple Independent Attackers

Several cryptographic routing protocols aim at ensuring the *route validity* in a network, *i.e.*, ensuring that the established route is an existing path in the network. However, flaws have been found in some protocols that are claimed secure. Cortier *et al.* [CDD12] show that when looking for attacks on properties such as route validity, it is sufficient to consider only five typologies with four nodes each. The authors assume cooperative attackers, *i.e.*, several attackers (compromised nodes) that share their knowledge using out-of-band resources or hidden channel. In this chapter, we extend their result to the case of non-cooperative attackers where several independent attackers that do not share their knowledge are considered. Such non-cooperative model is more realistic, particularly in the case of wireless sensor networks where nodes have low communication powers.

Contents

6.1 Introduction

Wireless ad-hoc networks have no existing infrastructure. This enables them to play a more and more important role in extending the coverage of traditional wireless infrastructure (*e.g.*, cellular networks, wireless LAN, etc.). These networks have no central administration control, and thus the presence of dynamic and adaptive routing protocol is necessary for them to work properly. Routing protocols aim to establish a route between distant nodes, enabling wireless nodes to communicate with the nodes that are outside their transmission range. As an example of routing protocols, we describe the Secure Routing Protocol SRP [PH02] applied on the Dynamic Source Routing protocol DSR [JMB01]. SRP is actually not a routing protocol by itself. However, it can be applied to an on-demand source routing protocol, such as DSR, in order to obtain a secure variant of this protocol. DSR is a protocol which is used when the source node (S) wants to communicate with another non-neighbor node, the destination node (D). DSR consists of two phases: the request phase, and the reply phase. After applying SRP, the two phases are as follows (assuming that each node already knows its neighbors).

- Request phase: The source node S broadcasts to its neighbors the request message: $\langle req, S, D, Id, [S], hmac(\langle req, S, D, Id\rangle, K_{SD})\rangle$, where req is an identifier indicating the request phase, $[S]$ is the initial route list, $hmac$ is a message authentication code (MAC) function, and K_{SD} is a symmetric key shared between S and D. Then, each intermediate node that receives the request, locally checks the route list: if the last (added) node in the list is one of its neighbors, then it broadcasts the request after appending its name (identifier) to the list, and drops the request message otherwise.

- Reply phase: Once the request reaches the destination D, it checks that the last node in the route is one of its neighbors, and that the MAC is correct. Then, D constructs a route reply, in particular it computes a new $hmac$ over the route accumulated in the request message and sends the following message back over the network: $\langle rep, D, S, Id, l, hmac(\langle rep, D, S, Id, l\rangle, K_{SD})\rangle$, where rep is an identifier indicating the reply phase, l is the obtained route list. Then, each intermediate node checks if its name appears in the route list between two of its neighbors, if so it forwards the message to the next node. Once the source S receives a message containing a route to D. Before accepting, it checks that the MAC is correct, that

the route does not contain loops, and that its neighbor in the route is indeed one of its neighbor in the network.

Security of routing protocols is a crucial for wireless ad-hoc networks since attacking routing protocol may disable the whole network operation. For example, forcing two nodes to believe in an invalid route (*i.e.*, a route that does not represent a real path in the network) will prevent them from communicating with each other. Several routing protocols [PH02, HPJ05, SDL+02] have been proposed to provide more guarantees on the resulting route in ad-hoc networks. However, they may still be subject to attacks. For example, a flaw has been discovered on SRP when applied to DSR allowing an attacker to modify the route [BV04], thus forcing the source node to believe in an invalid route. Another attack has also been found on the Ariadne protocol [HPJ05] in the same paper. This shows that, like other cryptographic protocols, designing routing protocols is a difficult and error-prone task. Thus, they have to be formally verified before they are used. However, decision procedures (*e.g.*, [Bla01, RT01]) that are applied on standard protocols such as confidentiality and secrecy, and tools such as AVISPA, ProVerif, and Scyther are not well-adapted to the case of routing protocols. A main reason for this is that, in contrast to standard security protocols where the attacker is assumed to control all the communications, an attacker for routing protocols has to be localized somewhere in the network. Thus, it can only control a finite number of nodes, and only listen to the messages sent by the nodes that are neighbors to the compromised nodes.

In 2010, Arnaud *et al.* [ACD10] proposed an NP decision procedure for analyzing routing protocols looking for attacks on route validity. However, in case of routing protocols the existence of an attack strongly depends on the network topology, *i.e.*, how nodes are connected and where *malicious* (compromised) nodes are located. This results in an infinite number of network topologies to verify, which is not tractable. Later in 2012, Cortier *et al.* [CDD12] proposed a reduction proof when looking for route validity property in presence of cooperative attackers that share their knowledge using out-of-band resources. The applied reduction technique results in only five topologies with four nodes each. However, it is not always realistic to consider cooperative attackers as this may results in an unpractical attacks. As an example, consider the case of ad-hoc sensors that are thrown from a plane into the enemy field during a battle. Indeed, due to their minimal configuration and quick deployment ad-hoc networks are suitable for emergency situations like natural disasters or military conflicts where no infrastructure is available. Thus, usually it is difficult to have out-of-band resources between malicious nodes. Note that, using hidden channels (*e.g.*, running other instances of routing protocol and using the obtained route) is unfeasible in some cases where nodes have low power capabilities (*e.g.*, sensor networks).

6.1.1 Contributions

In this chapter, we consider route validity property in presence of multiple independent (non-cooperative) attackers, *i.e.*, attackers that can not share their knowledge between each other. Then, we show that the reduction result obtained by Cortier *et al.* in [CDD12] in presence of cooperative attackers is also applicable to the case of the independent attackers. As the original paper the reduction proof consists of two steps: firstly, we show that if there is an attack on a routing protocol in a certain topology, then there is an attack on this protocol in a smaller topology obtained from the original topology by a simple reduction. Secondly, we show that applying the reduction procedure to any topology leads to (at most) five small topologies. We use the process calculus and the transition rules (after updating them to handle the behavior of the non-cooperative malicious nodes instead of the cooperative ones) presented in [CDD12], which are adapted to model routing protocols. Finally, we prove that a protocol satisfies the route validity property in presence of cooperative attackers in every network topology, if and only if, it satisfies the route validity in presence of non-cooperative attackers in every network topology.

6.1.2 Related work

Multiple attackers that do not share their knowledge are already considered in [ACRT11] to analyze the web-service applications looking for attacks that exploit XML format. Verifying protocols (for properties such as route validity) in presence of multiple independent attacker is equivalent to satisfiability of general constraints where knowledge monotonicity does not hold. The satisfiability of such kind of constraints has been proven to be NP-complete [Maz05, ACRT11]. However, in the case of routing protocols an infinite number of topologies have to be considered.

Some case studies and formal techniques dedicated for analyzing routing protocols have been proposed. For example, Ács *et al.* [ÁBV06] have developed a framework for analyzing on-demand source routing protocols. Benetti *et al.* [BMV10] have analyzed the ARAN [SLD+05] and endairA [ÁBV06] protocols with the AVISPA tool, and have discovered three attacks on ARAN protocol. Nanz and Hankin [NH06a] have proposed a formal framework for analyzing routing protocols of mobile wireless networks, and have shown how to automatically analyze a finite number of attack scenarios. Arnaud *et al.* [ACD10] have proposed an NP decision procedure for analyzing routing protocols for a finite number of sessions.

Few other works have presented reduction results in context of routing protocols. The first such approach was proposed by Andel *et al.* [ABY11]. They have proposed an approach to reduce the number of network topologies one has to consider, taking advantage of the symmetries. However, the total number of networks is still large even

for a bounded number of nodes. Recently, Cortier *et al.* [CDD12] have shown that only five topologies (four nodes each) need to be considered when looking for attacks on route validity in presence of attackers that share their knowledge. We show that the reduction proof presented in [CDD12] can be extended to the case of multiple independent attackers that do not share their knowledge. Note that, none of the previously mentioned work on routing protocols consider independent attackers.

6.1.3 Outline

In Section 6.2, we recall the syntax of the process calculus we use, and show how a routing protocol can be modeled using this calculus. We define route validity property and the corresponding attack in Section 6.3. Then in Section 6.4, we present a reduction proof and show that only five topologies are sufficient. Next in Section 6.5, we show that, for every network, a protocol satisfies route validity in presence of cooperative attackers, if and only if, it satisfies route validity in presence of non-cooperative attackers. Finally, we conclude in Section 6.6.

6.2 Modelling Routing Protocols

The Applied π-Calculus we used in previous chapters to model standard cryptographic protocols, and other calculi that have been proposed for the same purpose (*e.g.*, [AG97, MPW92]) are not well-adapted for routing protocols. For instance, in routing protocols the nodes have to perform some specific checks (*e.g.*, checking that a route is loop free, or that two nodes are neighbors) that cannot be easily modeled in such calculi. Moreover, network topology and broadcast communications have to be taken into account. A process calculus that is adapted to model routing protocol (and which follows the Applied π-Calculus) is presented in [CDD12] [1]. In this section, we recall the syntax of this calculus, then we adapt the related execution model to the case of multiple independent attackers (instead of attackers that share their knowledge).

6.2.1 Syntax

Similar to Applied π-Calculus, terms and function symbols are respectively used to represent messages and cryptographic primitives. Terms are names, variables, or function symbols applied to names and variables. An infinite set of names $\mathcal{N}$ and an infinite set of variables $\mathcal{X}$ are considered. A term is *ground* if it does not contain any variable. Each function symbol has a certain arity, and can only applied to terms of specific types (sorts).

[1] Note that, this calculus generalize the calculus give in [ACD10], which is is inspired by CBS# [NH06b], by allowing processes to perform any operation on the terms they receive and considering an arbitrary signature of function symbols.

Two special types are assumed: the type Agent that only contains agent's names and variables, and the type Term that subsumes all other types so that any term is of the type Term.

Example 6.1. (Signature [CDD12]). *A typical signature for representing the primitives used in SRP protocol is the signature* $\Sigma_{SRP} = \{hmac(\cdot,\cdot), \langle\cdot,\cdot\rangle, \cdot :: \cdot, [], req, rep\}$, *where* req *and* rep *are unitary constants identify the request and response phases respectively,* $[]$ *represents an empty list and other symbols are defined as follows:*

$$\langle\cdot,\cdot\rangle \ : \mathsf{Term} \times \mathsf{Term} \to \mathsf{Term} \qquad \cdot :: \cdot : \mathsf{Agent} \times \mathsf{List} \to \mathsf{List}$$
$$hmac(\cdot,\cdot) \ : \mathsf{Term} \times \mathsf{Term} \to \mathsf{Term}$$

The symbol $hmac(\cdot,\cdot)$ *takes two terms and computes the message authentication code of the first term with the second one as a key. The operator* $\langle\cdot,\cdot\rangle$ *produces a concatenation of two terms, and the operator* $\cdot :: \cdot$ *is the list constructor. We write* $\langle t_1, t_2, t_3\rangle$ *for the term* $\langle\langle t_1, t_2\rangle, t_3\rangle$ *, and* $[t_1, t_2, t_3]$ *for* $(([] :: t_1) :: t_2) :: t_3$.

Substitutions (σ) are well-sorted, and cycle-free. A substitution σ is extended to a homomorphism on functions, processes and terms as expected. Two terms t and s are *unifiable* if there exists a substitution θ, called *unifier*, such that $\theta(t) = \theta(s)$. The *most general unifier* (for short *mgu*) of two terms t and s is a unifier, denoted $mgu(t, s)$, such that for any unifier θ of t and s there exists a substitution σ with $\theta = \sigma \circ mgu(t, s)$ where $\circ$ is a composition of two mappings. We write $mgu(t, s) = \perp$ when t and s are not unifiable.

The calculus is parameterized by a set $\mathbf{P}$ of predicates to represent the checks performed by the agents, and a set $\mathcal{F}$ of functions over terms to represent the computations performed by the agents. The set of functions $\mathcal{F}$ contains functions that are more complex than basic cryptographic primitives represented by Σ, for example a function $f \ : \ (x, y, z) \mapsto hmac(\langle x, y\rangle, z)$ which takes three terms, concatenates the first two and then computes the MAC over them with the third term. They can also be used to model operations on lists, for example we can define a function that takes a list $[A_1, \ldots, A_n]$ and returns its reverse $[A_n, \ldots, A_1]$.

The intended behavior of each node in the network can be modeled by a process defined using the grammar given in Figure 6.1. The process $out(f(t_1, \ldots, t_n)).P$ computes the term $t = f(t_1, \ldots, t_n)$, emits t, and then behaves like P. The reception process $in(t).P$ expects a message m matching the pattern t and then behaves like $\sigma(P)$ where $\sigma = mgu(m, t)$. The process "$if\ \Phi\ then\ P$" behaves like P if Φ is true. Two processes P and Q running in parallel represented by the process $P|Q$. The replication process $!P$ denotes an infinite number of copies of P, all running in parallel. The process $new\ m.P$ creates a fresh name m and then behaves like P. Sometimes, for the sake of clarity we omit the null process. A *ground process* P is a process that have no free variables (we

$P, Q ::=$	Processes
0	null process.
$out(f(t_1, \ldots, t_n)).P$	emission
$in(t).P$	reception
$if\ \Phi\ then\ P$	conditional
$P \mid Q$	parallel composition.
$!P$	replication
$new\ m.P$	fresh name generation

where $t, t_1, \ldots, t_n$ are terms, m is a name, $f \in \mathcal{F}$, and Φ is a formula:

$\Phi, \Phi_1, \Phi_2 ::=$	Formula
$p(t'_1, \ldots, t'_n)$	$p \in \mathbf{P}$, $t'_1, \ldots, t'_n$ are terms
$\Phi_1 \wedge \Phi_2$	conjunction

FIGURE 6.1: Processes grammar

denote free variables of P by $fv(P)$), and a *parameterized process* $P(x_1, \ldots, x_n)$ is a process that has n variables $x_1, \ldots, x_n$ of type Agent, and such that $fv(P) \subseteq \{x_1, \ldots, x_n\}$. A *routing role* is a parameterized process that does not contain any name of type Agent. A routing protocol is then simply a set of routing roles.

The predicates $p \in \mathbf{P}$ are given together with their semantics that may depend on the underlying graph G that models the topology of the network. Such a graph $G = (E, V)$ is given by a set of vertices $V \subseteq \{A \in \mathcal{N} \mid A \text{ of type Agent}\}$, and a set of edges $E \subseteq V \times V$ where E is a reflexive and symmetric relation. There are two kinds of predicates: a set $\mathbf{P_I}$ of predicates whose semantics is independent of the graph and a set $\mathbf{P_D}$ of predicates whose semantics is dependent on the graph. For a graph dependent formula Φ and a graph G, $[\![\Phi]\!]_G = true$ (resp. $false$) denotes that Φ is $true$ (resp. $false$) in G.

Example 6.2. (Predicates [CDD12]). *For example, one can use the predicates* $\mathbf{P_{SRP}} = \mathbf{P_I} \cup \mathbf{P_D}$ *for SRP, with* $\mathbf{P_I} = \{\mathsf{checksrc}, \mathsf{checkdest}\}$ *and* $\mathbf{P_D} = \{\mathsf{check}, \mathsf{checkl}\}$. *The purpose of the* $\mathbf{P_I}$ *predicates is to model some checks that are performed by the source when it receives the route. The semantics of these predicates is defined as follows:*

- $\mathsf{checksrc}(S, l) = true$, *if and only if,* l *is of type* List *and its first element is* S,
- $\mathsf{checkdest}(D, l) = true$, *if and only if,* l *is of type* List *and its last element is* D *(i.e.,* $l = l_1 :: D$ *where* l_1 *is of type* List*).*

The predicates $\mathsf{checksrc}(S, l)$ *and* $\mathsf{checkdest}(D, l)$ *are used by the source process to verify that the first and last nodes of the established route are the source and destination of the route discovery respectively. While, the purpose of the* $\mathbf{P_D}$ *predicates is to model neighborhood checks. Given a graph* $G = (V, E)$*, their semantics is defined as follows:*

- $[\![\mathsf{check}(A, B)]\!]_G = true$*, if and only if,* $(A, B) \in E$ *or* $(B, A) \in E$*,*
- $[\![\mathsf{checkl}(C, l)]\!]_G = true$*, if and only if,* C *appears in* l*, and for any* l' *subterm of* l *we have* $(A, C) \in E$ *if* $l' = l_1 :: A :: C$ *and* $(C, B) \in E$ *if* $l' = l_1 :: C :: B$*.*

The aim of the predicate $[\![\mathsf{check}(A, B)]\!]_G$ *is to check if* A *and* B *are neighbors in* G*, while the aim of the predicate* $[\![\mathsf{checkl}(C, l)]\!]_G$ *is to check if the node* C *appears in* l *between two neighbors in* G*. We assume that each node knows its neighbors in the network, this can be achieved by running a certain neighbor discovery protocol in advance.*

The secure routing protocol SRP applied on DSR has been modeled in [CDD12]. Here we give only the source process as an example.

Example 6.3. (SRP Source Process [CDD12]). *Considering the signature* Σ_{SRP} *and the predicates* $\mathbf{P_{SRP}}$ *introduced before, and the set* $\mathcal{F}_{SRP}$ *of functions over terms that only contains the identity function (omitted for sake of clarity), the process played by the source* x_S *initiating the search of a route towards a destination* x_D *is given as follows:*

$$P_{src}(x_S, x_D) = new\ id.out(u_1).in(u_2).if\ \Phi_S\ then\ 0$$

where id *is a constant that identifies the request,* x_S *and* x_D *are variables of type* Agent*,* x_L *is a variable of type* List*, and* u_1*,* u_2 *and* Φ_S *are defined as follows:*

$$\begin{aligned} u_1 &= \langle req, x_S, x_D, id, [] :: x_S, hmac(\langle req, x_S, x_D, id\rangle, k_{x_S x_D})\rangle \\ u_2 &= \langle req, x_D, x_S, id, x_L, hmac(\langle req, x_D, x_S, id, x_L\rangle, k_{x_S x_D})\rangle \\ \Phi_S &= \mathsf{checkl}(x_S, x_L) \wedge \mathsf{checksrc}(x_S, x_L) \wedge \mathsf{checkdest}(x_D, x_L) \end{aligned}$$

6.2.2 Attacker Capabilities

We consider multiple independent attackers that do not have the ability to share knowledge between each other using any out-of-band-resources or hidden channels. Each attacker compromises a node in the network, and thus can only listen and send messages to its neighbors. We assume that the capabilities of each attacker are modeled by a deduction relation $\vdash$. The relation $I \vdash t$ means that the term t is *deducible* from the set of terms I. Such a deduction relation can be defined through an inference system, *i.e.*, a finite set of rules of the form $\dfrac{t_1 \cdots t_n}{t}$, where $t, t_1, \ldots, t_n$ are of type Term. It follows that $I \vdash t$ if there exists a proof tree with root t and leaves in I.

Example 6.4. (Inference System [CDD12]). *One can associate to the SRP signature* Σ_{SRP}*, the following inference system:*

$$\frac{t \in I}{I \vdash t} \qquad \frac{I \vdash t_1 \quad I \vdash t_2}{I \vdash \langle t_1, t_2 \rangle} \qquad \frac{I \vdash \langle t_1, t_2 \rangle}{I \vdash t_i}\ i \in \{1,2\}$$

$$\frac{I \vdash l_1 \quad I \vdash l_2}{I \vdash l_1 :: l_2} \qquad \frac{l_1 :: l_2}{l_i}\ i \in \{1,2\} \qquad \frac{I \vdash t_1 \quad I \vdash t_2}{I \vdash hmac(t_1, t_2)}$$

where I *is the knowledge of the attacker,* t*,* t_1 *and* t_2 *are variables of type* Term*,* l_1 *is a variable of type* List*, and* l_2 *is a variable of type* Agent*. The system gives the attacker an ability to concatenate terms, build lists, as well as to retrieve their components. The first rule is an axiom, and the last rule models the fact that the attacker can also compute a MAC provided he knows the corresponding key.*

6.2.3 Configuration and Topology

In the classical Dolev-Yao model [DY83b], the attacker controls all the communication channels. It is not the case with ad-hoc networks where the attacker has to be located at a specific node, and thus can only interact with its neighbours. Similarly to [CDD12], we represent a network *topology* by a tuple $\mathcal{T} = (G, \mathcal{M}, S, D)$ where:

- $G = (V, E)$ is an undirected graph. The set $V \subseteq \{A \in \Sigma_0 \mid A \text{ of sort } \mathsf{Agent}\}$ represents the set of network nodes. An edge (A, B) in E means that the two nodes A and B are neighbours. We assume that a node can receive messages that it sent, that is for any $A \in V$, we have that $(A, A) \in E$;
- $\mathcal{M} = \{M_i\}_{i=1}^{i=k}$ is a set of nodes that are controlled by k independent attackers we have in the network, where each attacker controls only one node. Note that $\mathcal{M} \subseteq V$. These nodes that are in $\mathcal{M}$ are called malicious whereas nodes not in $\mathcal{M}$ are called honest;
- S and D are respectively the source and the destination nodes of the routing protocol run. We assume that S and D are always honest.

Note that malicious nodes cannot communicate using out-of-band resources or hidden channels.

Example 6.5. (Topology). *A possible topology* $\mathcal{T}_0 = (G_0, \mathcal{M}_0, S, D)$ *is depicted in Figure 6.2, where* S (→○) *is the source,* D (○→) *is the destination,* A *is a honest node (colored in white), and* M_1 *and* M_2 *are two malicious nodes (colored in black), i.e.,* $\mathcal{M}_0 = \{M_1, M_2\}$*.*

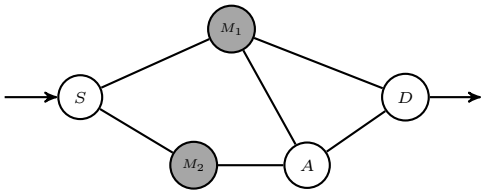


FIGURE 6.2: Topology $\mathcal{T}_0$

A *configuration* of a network is a pair specifying the process to be executed by each node, and the initial knowledge of each attacker (malicious node).

Definition 6.1. (Configuration [CDD12]). *A* configuration *of a network is a pair* $(\mathcal{P}, \mathcal{I})$ *where:*

- $\mathcal{P} = \{\lfloor P \rfloor_A$ *for some* $A \in V\}$ *is a multiset with* $\lfloor P \rfloor_A$ *represents the process* P *that is executed by the node* A*. In the following, we may write* $\lfloor P \rfloor_A$ *instead of* $\{\lfloor P \rfloor_A\}$*;*
- *We assume an independent knowledge for each attacker as we define the set of sets of terms* $\mathcal{I} = \{I_i\}_{i=1}^{i=k}$*, where the set* I_i *represents the messages that the malicious node* $M_i \in \mathcal{M}$ *has initially and also that it has observed on the network.*

Example 6.6. (Configuration [CDD12]). *A typical initial configuration for the SRP protocol is:* $K = (\lfloor P_{src}(S, D) \rfloor_S \mid \lfloor P_{req}(A) \rfloor_A \mid \lfloor P_{rep}(A) \rfloor_A \mid \lfloor P_{dest}(D) \rfloor_D; \mathcal{I})$.

6.2.4 Execution Model

Figure 6.3 presents the set of communication rules for a given topology $\mathcal{T} = (G, \mathcal{M}, S, D)$ with $G = (V, E)$. They are similar to the ones used in [ACD10, CDD12] with the differences that we use a separate set of knowledge for each attacker, and we assume that the message sent by a certain malicious node can be captured by its malicious neighbors due to broadcast nature of the communications in wireless add-hoc networks (rule IN). Thanks to COMM rule, neighbour nodes can exchange messages between each others. When a node sent a message, it is added to the knowledge I_i if this node is a neighbour of M_i. A malicious node M_i is provided the capability to send any message, it can build from its knowledge I_i, to one of its neighbours. This is modeled using IN rule. Note that, like in case of COMM rule the sent message is captured by neighbour malicious nodes. The rule IF-THEN states that a node A executes the process P only if the formula Φ is true. PAR rule says that parallel processes are equivalent to parallel nodes running these processes. The replication process $!P$ is expanded using the rule REPL. The last rule NEW says that nodes can use fresh names of their choice when required. The relation $\rightarrow_{\mathcal{T}}^*$ is the reflexive and transitive closure of $\rightarrow_{\mathcal{T}}$.

COMM $(\{\lfloor in(t'_j).P_j \rfloor_{A_j} \mid mgu(t,t'_j) \neq \bot, (A,A_j) \in E\} \cup \lfloor out(f(t_1,\ldots,t_n).P \rfloor_A \cup \mathcal{P};\mathcal{I}) \rightarrow_{\mathcal{T}} (\lfloor P_j\sigma_j \rfloor_{A_j} \cup \lfloor P \rfloor_A \cup \mathcal{P};\mathcal{I}')$
where $\sigma_j = mgu(t,t'_j)$ with $t = f(t_1,\ldots,t_n)$, and for $i \in \{1,\ldots,k\}$, if $(A,M_i) \in E$ then $I'_i = I_i \cup \{t\}$ else $I'_i = I_i$.

IN $(\lfloor in(t').P \rfloor_A \cup \mathcal{P};\mathcal{I}) \rightarrow_{\mathcal{T}} (\lfloor P\sigma \rfloor_A \cup \mathcal{P};\mathcal{I}')$ if $(A,M_j) \in E$, $I_j \vdash t$ and $M_j \in \mathcal{M}$
where $\sigma = mgu(t,t')$, and if $(M_j,M_i) \in E$ then $I'_i = I_i \cup \{t\}$ else $I'_i = I_i$.

IF-THEN $(\lfloor \text{if } \Phi \text{ then } P \rfloor_A \cup \mathcal{P};\mathcal{I}) \rightarrow_{\mathcal{T}} (\lfloor P \rfloor_A \cup \mathcal{P};\mathcal{I})$ if $[\![\Phi]\!]_G = true$.

PAR $(\lfloor P_1|P_2 \rfloor_A \cup \mathcal{P};\mathcal{I}) \rightarrow_{\mathcal{T}} (\lfloor P_1 \rfloor_A \cup \lfloor P_2 \rfloor_A \cup \mathcal{P};\mathcal{I})$

REPL $(\lfloor !P \rfloor_A \cup \mathcal{P};\mathcal{I}) \rightarrow_{\mathcal{T}} (\lfloor P \rfloor_A \cup \lfloor !P \rfloor_A \cup \mathcal{P};\mathcal{I})$

NEW $(\lfloor new\ m.P \rfloor_A \cup \mathcal{P};\mathcal{I}) \rightarrow_{\mathcal{T}} (\lfloor P\{m \mapsto m'\rfloor_A\} \cup \mathcal{P};\mathcal{I})$
where m' is a fresh name.

FIGURE 6.3: Transition system.

6.3 Route Validity Property

A routing protocol satisfies route validity property if it results in a *valid route*. In this section, we define the valid route and the attack on a routing protocol. Note that, we do not consider the case of *wormhole attack*, so a route that contains where two successive non-neighboring malicious nodes is considered a valid route.

Definition 6.2. (Valid route [CDD12]). *Let $\mathcal{T} = (G,\mathcal{M},S,D)$ be a topology with $G = (V,E)$, we say that a list $l = [A_1,\ldots,A_n]$ of agent names is a valid route in $\mathcal{T}$, if and only if, for any $i \in \{1,\ldots,n-1\}$, we have that $(A_i,A_{i+1}) \in E$ or $A_i,A_{i+1} \in \mathcal{M}$.*

Similarly to [CDD12], we assume that an instance of a routing protocol contains a process that outputs the term $end(l)$ when the route, represented by list l, is established. This allows us to represent the route validity property as a reachability property. The attack on the configuration of a routing protocol is defined as follows.

Definition 6.3. (Attack on a configuration K in $\mathcal{T}$ [CDD12]). *Let $\mathcal{T} = (G,\mathcal{M},S,D)$ be a topology and K be a configuration. We say that K admits an* attack *in $\mathcal{T}$ if $K \rightarrow_{\mathcal{T}}^{*} (\lfloor out(end(l)).P \rfloor_A \cup \mathcal{P};\mathcal{I})$ for some A, P, $\mathcal{P}$, $\mathcal{I}$, and some term l that is not a valid route in $\mathcal{T}$.*

For a given routing protocol $\mathcal{P}_{routing}$, we only consider configurations that are made up using a routing role P_0 and roles of the protocol, such that P_0 is the only process that contains a special action $out(end(l))$. Moreover, we check whether the security property holds when the source and the destination are honest. Note that, we consider the case where an honest source initiates a session with a malicious node ($P_{src}(S, M)$ for $M \in \mathcal{M}$ can occur in the configuration).

Definition 6.4. (Configuration valid for $\mathcal{P}_{routing}$ and P_0 [CDD12]). *Let $\mathcal{P}_{routing}$ be a routing protocol, and P_0 be a routing role. Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$, and $\mathcal{I}$ be a set of sets representing the initial knowledge of the malicious nodes in $\mathcal{M}$. A configuration $K = (\mathcal{P}, \mathcal{I})$ is valid for $\mathcal{P}_{routing}$ and P_0 with respect to $\mathcal{T}$ and $\mathcal{I}$ if*

- *$\mathcal{P}$ is of the form $\lfloor P_0(S, D) \rfloor_S \uplus \mathcal{P}'$; and*
- *For every $\lfloor P' \rfloor_{A_1} \in \mathcal{P}'$, we have that $P' = P(A_1, \ldots, A_n)$ for some process $P(x_1, \ldots, x_n) \in \mathcal{P}_{routing}$ and nodes $A_2, \ldots, A_n \in V$; and*
- *$P_0(S, D)$ is the only process that contains an action like $out(end(l))$.*

The attack on a routing protocol $\mathcal{P}_{routing}$ and a routing role P_0 with respect a topology $\mathcal{T}$ is defined as follows.

Definition 6.5. (Attack on $\mathcal{P}_{routing}$ and P_0 w.r.to $\mathcal{I}$ [CDD12]). *Let $\mathcal{P}_{routing}$ be a routing protocol, and P_0 be a routing role. Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology, and $\mathcal{I}$ be a set which represents the initial knowledge. We say that there is an attack on $\mathcal{P}_{routing}$ and P_0 in $\mathcal{T}$ with respect to $\mathcal{I}$ if, there exists a configuration K that is valid for $\mathcal{P}_{routing}$ and P_0 with respect to $\mathcal{T}$ and $\mathcal{I}$ such that K admits an attack in $\mathcal{T}$.*

6.4 Reduction Procedure

In this section, we present a reduction procedure, and we show that if there is an attack on route validity in a given topology then there is an attack in a smaller topology obtained by applying the reduction procedure to the initial one. Hence, applying the reduction to a certain topology allows us to verify a protocol in a smaller topology. We show that applying the reduction procedure on any topology results in one of five topologies, each consists of only four nodes. Then, we show that considering only five topologies is sufficient when looking for attacks on route validity. Following [CDD12], our reduction procedure consists of two main steps:

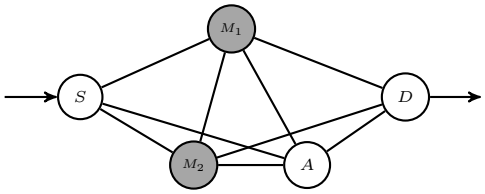
1. Adding edges to the graph yielding a *quasi-complete* topology.
2. Merging nodes that have the same nature (honest or malicious) and same neighbors.

6.4.1 From an Arbitrary Topology to a Quasi-Complete One

In order to merge nodes while preserving the attack, they must have the same nature and same neighbors. To ensure that most of the nodes have the same neighbors we first add edges to the graph. Actually, we add all edges except one. We show that the attack is preserved when we add these edges.

Definition 6.6. (Quasi-completion [CDD12]). *Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$, and let A, B be two nodes in V that are not both malicious and such that $(A, B) \notin E$. The quasi-completion of $\mathcal{T}$ with respect to (A, B) is a topology $\mathcal{T}^+ = (G^+, \mathcal{M}, S, D)$ such that $G^+ = (V, E^+)$ with $E^+ = V \times V \setminus \{(A, B), (B, A)\}$.*

Example 6.7. (Quasi-completion). *A possible quasi-completion $\mathcal{T}_0^+$ of the topology $\mathcal{T}_0$ shown in Figure 6.2 is the one with respect to the pair (S, D) given below. As we see the graph is almost highly connected, the only missing edge is (S, D).*



Note that, predicates have to be completion-friendly so that their values are preserved when adding some edges to the graph.

Definition 6.7. (Completion-friendly [CDD12]). *A predicate p is completion-friendly if $[\![p(t_1, \ldots, t_n)]\!]_G = true$ implies that $[\![p(t_1, \ldots, t_n)]\!]_{G^+} = true$ for any ground terms $t_1, \ldots, t_n$ and any quasi-completion $\mathcal{T}^+ = (G^+, \mathcal{M}, S, D)$ of $\mathcal{T} = (G, \mathcal{M}, S, D)$. We say that a routing protocol (resp. a configuration) is completion-friendly if the predicates $\mathbf{P_D}$, i.e., the predicates that are dependent of the graph are completion-friendly.*

Lemma 6.1. (Quasi-completion). *Let $\mathcal{T}$ be a topology, K_0 be a configuration that is completion-friendly. If there is an attack on K_0 in $\mathcal{T}$, then we can find two non-neighboring nodes $B, C \in V$ that are not both malicious and a topology $\mathcal{T}^+$ quasi-completion of $\mathcal{T}$ with respect to (B, C), such that there exists an attack on K_0 in $\mathcal{T}^+$.*

Proof. Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$ and K_0 be a configuration that is completion-friendly. Assume that there is an attack on K_0 in $\mathcal{T}$. Then, by the definition of the attack, there exist A, P, $\mathcal{P}$, $\mathcal{I}$ and $l_0 = [A_1, \ldots, A_n]$, such that $K_0 \rightarrow_{\mathcal{T}}^* (\lfloor out(end(l_0)).P \rfloor_A \cup \mathcal{P}; \mathcal{I}) = K$ and l_0 is not a valid route in $\mathcal{T}$, *i.e.*, there exists $1 \leq a \leq n$ such that $(A_a, A_{a+1}) \notin E$ and ($A_a \notin \mathcal{M}$ or $A_{a+1} \notin \mathcal{M}$). Let $\mathcal{T}^+ = (G^+, \mathcal{M}, S, D)$ be a quasi-completion of $\mathcal{T}$ with respect to (A_a, A_{a+1}). Then, we have that $G^+ = (V, E^+)$ with $E^+ = V \times V \setminus \{(A_a, A_{a+1}), (A_{a+1}, A_a)\}$. We show by

induction on the length r of a derivation $K_0 \rightarrow_{\mathcal{T}}^{r} K_r$[2] that K_r is completion-friendly and that $K_0 \rightarrow_{\mathcal{T}^+}^{r} K_r$. This will allow us to obtain that $K_0 \rightarrow_{\mathcal{T}^+}^{*} (\lfloor out(end(l_0)).P \rfloor_A \cup \mathcal{P}; \mathcal{I})$, and as by definition of $\mathcal{T}^+$, l_0 is not a valid route in $\mathcal{T}^+$ we conclude that K_0 admits an attack in $\mathcal{T}^+$.

For $r = 0$, since K_0 is completion-friendly, we can easily conclude. Now, assume that $K_0 \rightarrow_{\mathcal{T}}^{r-1} K_{r-1}$ then, by induction hypothesis, we have that K_{r-1} is completion-friendly and $K_0 \rightarrow_{\mathcal{T}^+}^{r-1} K_{r-1}$. Suppose that, $K_{r-1} \rightarrow_{\mathcal{T}} K_r$. Since K_{r-1} is completion-friendly, then K_r is Completion-friendly as there is no rule that introduce new predicates or functions. We show that $K_{r-1} \rightarrow_{\mathcal{T}^+} K_r$ by case analysis on the rule involved in the step $K_{r-1} \rightarrow_{\mathcal{T}} K_r$:

Case of the rule IF-THEN: we have that $K_{r-1} = (\lfloor \text{if } \Phi \text{ then } P \rfloor_A \cup \mathcal{P}; \mathcal{I})$, $K_r = (\lfloor P \rfloor_A \cup \mathcal{P}; \mathcal{I})$ and $[\![\Phi]\!]_G = true$. Since K_{r-1} is completion-friendly and $[\![\Phi]\!]_G = true$ then $[\![\Phi]\!]_{G^+} = true$, it follows that we can apply the rule IF-THEN on K_{r-1} in $\mathcal{T}^+$, and thus we get that $K_{r-1} \rightarrow_{\mathcal{T}^+} K_r$.

Case of the rule IN: in such a case, we have that $K_{r-1} = (\lfloor in(t').P \rfloor_A \cup \mathcal{P}; \mathcal{I})$, $K_r = (\lfloor P\sigma \rfloor_A \cup \mathcal{P}; \mathcal{I}')$ where $\sigma = mgu(t, t')$, $(A, M_j) \in E$ for some $M_j \in \mathcal{M}$, $I_j \vdash t$ and for $i \in \{1, \ldots, k\}$, if $(M_j, M_i) \in E$ then $I'_i = I_i \cup \{t\}$, else $I'_i = I_i$. We have that $E \subseteq E^+$, then $(A, M_j) \in E^+$ and if $(M_j, M_i) \in E$ then $(M_j, M_i) \in E^+$. Thus, we can conclude that $K_{r-1} \rightarrow_{\mathcal{T}^+} K_r$.

Rule COMM: we have that: $K_{r-1} = (\{\lfloor in(t'_j).P_j \rfloor_{A_j} \mid mgu(t, t'_j) \neq \bot, (A, A_j) \in E\} \cup \lfloor out(f(t_1, \ldots, t_n).P \rfloor_A \cup \mathcal{P}; \mathcal{I})$ and $K_r = (\lfloor P_j\sigma_j \rfloor_{A_j} \cup \lfloor P \rfloor_A \cup \mathcal{P}; \mathcal{I}')$ where $t = f(t_1, \ldots, t_n)$, $\sigma_j = mgu(t, t'_j)$, and for $i \in \{1, \ldots, k\}$, if $(A, M_i) \in E$ then $I'_i = I_i \cup \{t\}$, else $I'_i = I_i$. As $E \subseteq E^+$, then $(A, A_j) \in E$ implies that $(A, A_j) \in E^+$, and $(A, M_i) \in E$ implies that $(A, M_i) \in E^+$. Thus, we have that $K_{r-1} \rightarrow_{\mathcal{T}^+} K_r$.

Case of the rules PAR, REPL, and NEW: these rules do not depend on the underlying graph. This allows us to easily conclude. □

6.4.2 Reducing the Size of the Topology

In this step, we merge nodes that have the same nature and same neighbors. The initial knowledge of malicious nodes are joined when they are merged. In fact, sometimes one malicious node could do the job of several malicious nodes if we give it the required initial knowledge, for instance the case where we have a chain of malicious nodes. Also, in some cases existence or absence of some malicious nodes has no effect. We show that if there exists an attack in a given topology $\mathcal{T}$ then there exists an attack in a reduced topology $\rho(\mathcal{T})$ (some times written $\mathcal{T}\rho$) where ρ is a node *renaming mapping*.

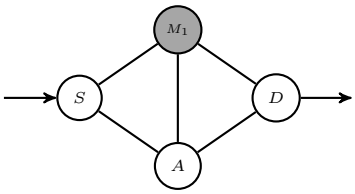
[2] $K_0 \rightarrow_{\mathcal{T}}^{r} K_r$ means that K_r is obtained from K_0 in r steps

Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$, and such that E is a reflexive and symmetric relation. Similarly to [CDD12], we say that a renaming $\rho : V \mapsto V$ on agent names:

- *preserves honesty* of $\mathcal{T}$ if, $\rho(A) \in \mathcal{M}$ if and only if $A \in \mathcal{M}$ for every $A \in V$.
- *preserves neighbourhood* of $\mathcal{T}$ if, $\rho(A) = \rho(B)$ implies that A and B have the same set of neighbours.

We use $t\rho$ to denote the application of the renaming ρ on term t. We extend this notion to to set of terms, configurations, graphs, and topologies. For example we denote by $G\rho$, with $G = (V, E)$, the graph $(V\rho, E')$ where $E' = \{(\rho(A), \rho(B))|(A, B) \in E\}$. Note that when we apply a renaming ρ to a configuration $K = (\mathcal{P}, \mathcal{I})$ then the knowledge $I_i \in \mathcal{I}$ of $M_i \in \mathcal{M}$ is joined with the knowledge $I_{i'}$ of $M_i\rho = M_{i'}$ and the I_i is removed from $\mathcal{I}$.

Example 6.8. (Renaming Mapping) *Consider the quasi-completion $\mathcal{T}_0^+$ we have seen in Example 6.7, a possible renaming ρ_0 that preserves neighborhood and honesty and that allows us to reduce the size of the graph is defined by: $\rho_0(S) = S, \rho_0(A) = A, \rho_0(M_1) = \rho_0(M_2) = M_1, \rho_0(D) = D$. The resulting topology $\mathcal{T}_0^+\rho_0$ is as follows:*



Here, the two malicious nodes M_1 and M_2 are merged in M_1 then the knowledge I_2 corresponding to M_2 should be pooled with I_1 (the knowledge of M_1). For instance, assume that we have initially $I_1 = \{M_1, S, D\}$, $I_2 = \{M_2, S, A\}$ and $\mathcal{I} = \{I_1, I_2\}$ then after merging we have that $I_1\rho_0 = \{M_1, S, D, A\}$ and $\mathcal{I}\rho_0 = \{I_1\rho_0\}$. Note that, ρ_0 does not preserve neighborhood of the topology $\mathcal{T}_0$ shown in Figure 6.2, which emphasizes the importance of the completion step in order to make a safe merging.

Definition 6.8. (Projection-friendly [CDD12]). *A predicate p is projection-friendly if $[\![p(t_1, \ldots, t_n)]\!]_G = true$ implies $[\![p(t_1\rho, \ldots, t_n\rho)]\!]_{G\rho} = true$ for any ground terms $t_1, \ldots, t_n$ and any renaming ρ that preserves neighborhood and honesty. A function f over terms is projection-friendly if $f(t_1\rho, \ldots, t_n\rho)) = f(t_1, \ldots, t_n)\rho$ for any ground terms $t_1, \ldots, t_n$ and any renaming ρ that preserves neighborhood and honesty. We say that a routing protocol (resp. a configuration) is projection-friendly if the predicates $\mathbf{P_I} \cup \mathbf{P_D}$ and the functions in $\mathcal{F}$ are projection-friendly.*

Lemma 6.2. (Reducing). *Let $\mathcal{T}$ be a topology, K_0 be a configuration that is projection-friendly, and ρ be a renaming that preserves neighborhood and honesty in $\mathcal{T}$. If there is an attack on K_0 in $\mathcal{T}$, then there exists an attack on K_0' in $\mathcal{T}'$ where K_0' and $\mathcal{T}'$ are obtained by applying ρ on K_0 and $\mathcal{T}$ respectively.*

Proof. Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$ and K_0 be a configuration that is projection-friendly. Assume that there is an attack on K_0 in $\mathcal{T}$. Then, by the definition of the attack, there exist A, P, $\mathcal{P}$, $\mathcal{I}$ and $l_0 = [A_1, \ldots, A_n]$, such that $K_0 \rightarrow^*_{\mathcal{T}} K = (\lfloor out(end(l_0)).P \rfloor_A \cup \mathcal{P}; \mathcal{I})$ and l_0 is not a valid route in $\mathcal{T}$. Let $K'_0 = K_0\rho$ and $\mathcal{T}' = \mathcal{T}\rho = (G\rho, \mathcal{M}\rho, S\rho, D\rho)$ where $G\rho = (V\rho, E\rho)$. We show by induction on the length r of a derivation $K_0 \rightarrow^r_{\mathcal{T}} K_r$ that K_r is projection-friendly and $K'_0 \rightarrow^r_{\mathcal{T}'} K'_r$ with $K'_r = K_r\rho$. This will allow us to obtain that $K'_0 \rightarrow^*_{\mathcal{T}'} K'$ with $K' = K\rho$.

For $r = 0$, since $K'_0 = K_0\rho$ and K_0 is projection-friendly, we can easily conclude. Assume that $K_0 \rightarrow^{r-1}_{\mathcal{T}} K_{r-1}$, then, by induction hypothesis, we have that K_{r-1} is projection-friendly and $K'_0 \rightarrow^{r-1}_{\mathcal{T}'} K'_{r-1}$ with $K'_{r-1} = K_{r-1}\rho$. Suppose that, $K_{r-1} \rightarrow_{\mathcal{T}} K_r$. Since K_{r-1} is projection-friendly, then K_r is projection-friendly as there is no rule that introduce new predicates or functions. We show that $K'_{r-1} \rightarrow_{\mathcal{T}'} K'_r$ with $K'_r = K_r\rho$ by case analysis on the rule involved in the step $K_{r-1} \rightarrow_{\mathcal{T}} K_r$:

Case of the rule IF-THEN: we have that $K_{r-1} = (\lfloor \text{if } \Phi \text{ then } P \rfloor_A \cup \mathcal{P}; \mathcal{I})$, $K_r = (\lfloor P \rfloor_A \cup \mathcal{P}; \mathcal{I})$ and $[\![\Phi]\!]_G = true$. Since K_{r-1} is projection-friendly and $[\![\Phi]\!]_G = true$, then $[\![\Phi\rho]\!]_{G\rho} = true$, it follows that we can apply the rule IF-THEN on $K'_{r-1} = K_{r-1}\rho = (\lfloor \text{if } \Phi\rho \text{ then } P\rho \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}\rho)$, and thus we get that $K'_{r-1} \rightarrow_{\mathcal{T}'} K'_r$ with $K'_r = (\lfloor P\rho \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}\rho) = K_r\rho$.

Case of the rule IN: in such a case, we have that $K_{r-1} = (\lfloor in(t').P \rfloor_A \cup \mathcal{P}; \mathcal{I})$, $K_r = (\lfloor P\sigma \rfloor_A \cup \mathcal{P}; \mathcal{I}')$ where $\sigma = mgu(t, t')$, $(A, M_j) \in E$ for some $M_j \in \mathcal{M}$, $I_j \vdash t$, and for $i \in \{1, \ldots, k\}$, if $(M_j, M_i) \in E$ then $I'_i = I_i \cup \{t\}$, else $I'_i = I_i$. Furthermore, we have that $K'_{r-1} = K_{r-1}\rho = (\lfloor in(t'\rho).P\rho \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}\rho)$, $(A\rho, M_j\rho) \in E\rho$ since $(A, M_j) \in E$, $M_j\rho \in \mathcal{M}\rho$ since $M_j \in \mathcal{M}$ and ρ preserve honesty, $I_i\rho \vdash t\rho$, and if $(M_j, M_i) \in E$ for some $M_i \in \mathcal{M}$, then $(M_j\rho, M_i\rho) \in E\rho$ and $M_i\rho \in \mathcal{M}\rho$ since ρ preserves neighborhood and honesty. Thus, $K'_{r-1} \rightarrow_{\mathcal{T}'} (\lfloor (P\rho)\sigma' \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}'\rho) = K'_r$, where $\sigma' = mgu(t\rho, t'\rho)$. Note that, $(P\rho)\sigma' = (P\sigma)\rho$, and thus $K'_r = (\lfloor (P\sigma)\rho \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}'\rho) = K_r\rho$.

Case of the rule COMM: in this case $K_{r-1} = (\{\lfloor in(t'_j).P_j \rfloor_{A_j} \mid mgu(t, t'_j) \neq \bot, \text{ and } (A, A_j) \in E\} \cup \lfloor out(f(t_1, \ldots, t_n).P \rfloor_A \cup \mathcal{P}; \mathcal{I})$ and $K_r = (\lfloor P_j\sigma_j \rfloor_{A_j} \cup \lfloor P \rfloor_A \cup \mathcal{P}; \mathcal{I}')$ where $t = f(t_1, \ldots, t_n)$, $\sigma_j = mgu(t, t'_j)$, and for $i \in \{1, \ldots, k\}$, if $(A, M_i) \in E$ then $I'_i = I_i \cup \{t\}$, else $I'_i = I_i$. We have that, $K'_{r-1} = K_{r-1}\rho = (\{\lfloor in(t'_j\rho).P_j\rho \rfloor_{A_j\rho} \mid mgu(t\rho, t'_j\rho) \neq \bot, (A\rho, A_j\rho) \in E'\} \cup \lfloor out(f(t_1, \ldots, t_n)\rho).P\rho \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}\rho)$, $f(t_1, \ldots, t_n)\rho = f(t_1\rho, \ldots, t_n\rho)$ since f is projection-friendly, and $(A\rho, M_i\rho) \in E'$ if $(A, M_i) \in E$, then $K'_{r-1} \rightarrow_{\mathcal{T}'} (\lfloor (P_j\rho)\sigma'_j \rfloor_{A_j\rho} \cup \lfloor P\rho \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}'\rho) = K'_r$, where $\sigma'_j = mgu(t\rho, t'_j\rho)$. Thus, as $(P_j\rho)\sigma'_j = (P_j\sigma_j)\rho$, $K'_r = (\lfloor (P_j\sigma_j)\rho \rfloor_{A_j\rho} \cup \lfloor P\rho \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}'\rho) = K_r\rho$.

Case of the rules PAR, REPL, and NEW: these rules do not depend on the underlying graph. This allows us to easily conclude.

Hence, $K'_0 = K_0\rho \rightarrow^*_{\mathcal{T}'} (\lfloor out(end(l_0\rho)).P\rho \rfloor_{A\rho} \cup \mathcal{P}\rho; \mathcal{I}\rho) = K' = K\rho$. In order to conclude that there is an attack on K'_0 in $\mathcal{T}'$, it remains to show that $l_0\rho = [A_1\rho, \ldots, A_n\rho] = [A'_1, \ldots, A'_n]$ is not a valid route in $\mathcal{T}'$. First, we note that: (i) If $B \notin \mathcal{M}$ then $B\rho \notin \mathcal{M}\rho$. Assume that, $B \notin \mathcal{M}$ and $B\rho \in \mathcal{M}\rho$. Then there exists $C \in \mathcal{M}$ such that $B\rho = C\rho$. Thus, as ρ preserve honesty we have that B and C are both malicious or both honest, which leads to a contradiction. (ii) If $(B_1, B_2) \notin E$ then $(B_1\rho, B_2\rho) \notin E\rho$. Let $B_1, B_2 \in V$, $\rho(B_1) = D_1$ and $\rho(B_2) = D_2$ such that$(B_1, B_2) \notin E$. Suppose that $(D_1, D_2) = (B_1\rho, B_2\rho) \in E\rho$, then by definition of $E\rho$ there exist two nodes $C_1, C_2 \in V$ such that $\rho(C_1) = D_1$, $\rho(C_2) = D_2$ and $(C_1, C_2) \in E$. Since $\rho(B_1) = \rho(C_1) = D_1$ and ρ preserves neighborhood, we get that $N_G(B_1) = N_G(C_1)$. It follows that B_1 and C_2 are neighbors in G since C_1 and C_2 are neighbors in G. Similarly, we have that $N_G(B_2) = N_G(C_2)$. Thus B_1 and B_2 are also neighbors in G, *i.e.*, $(B_1, B_2) \in E$ which leads to a contradiction. Hence, $(B_1\rho, B_2\rho) \notin E\rho$.

Now, as l_0 is not a valid route in $\mathcal{T}$, there exists $1 \leq a \leq n$ such that $(A_a, A_{a+1}) \notin E$ and ($A_a \notin \mathcal{M}$ or $A_{a+1} \notin \mathcal{M}$). Then, $(A_a\rho, A_{a+1}\rho) \notin E\rho$ and ($A_a\rho \notin \mathcal{M}\rho$ or $A_{a+1}\rho \notin \mathcal{M}\rho$). Hence, $l_0\rho$ is not a valid route in $\mathcal{T}'$ and we can conclude. □

6.4.3 Five Topologies are Sufficient

We show that for a routing protocol $\mathcal{P}_{routing}$ there is an attack on an arbitrary topology if and only if there is an attack on one of five particular topologies. Our result holds for an unbounded number of sessions since we consider arbitrarily many instances of the roles occurring in $\mathcal{P}_{routing}$.

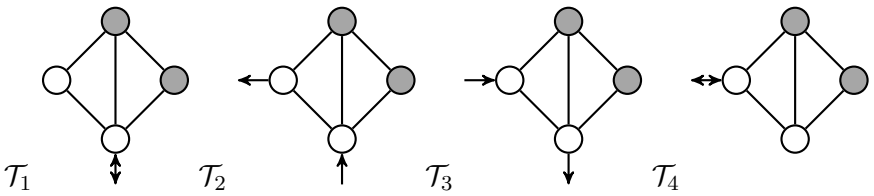
Theorem 6.1. (Five topologies). *Let $\mathcal{I}$ be a set of knowledge, and let $\mathcal{P}_{routing}$ be a routing protocol and P_0 be a routing role which are completion-friendly and projection-friendly. There exists a topology $\mathcal{T}$ such that there is an attack on $\mathcal{P}_{routing}$ and P_0 in $\mathcal{T}$ with respect to $\mathcal{I}$, if and only if, there are five particular topologies: $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$, $\mathcal{T}_4$ and $\mathcal{T}_5$, such that there is an attack on $\mathcal{P}_{routing}$ and P_0 with respect to $\mathcal{I}$ in one of them.*

Proof. If there is an attack on $\mathcal{P}_{routing}$ and P_0 with respect to $\mathcal{I}$ in one of the five particular topologies, we easily conclude that, there exists a topology $\mathcal{T}$ such that there is an attack on $\mathcal{P}_{routing}$ and P_0 in $\mathcal{T}$ with respect to $\mathcal{I}$. We consider now the other implication. Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$, $\mathcal{I}$ be a set of knowledge and $K = (\mathcal{P}, \mathcal{I})$ be a valid configuration for $\mathcal{P}_{routing}$ and P_0 with respect to $\mathcal{T}$, such that there is an attack on K in $\mathcal{T}$. Without lost of generality, we assume that V contains at least three distinct honest nodes and two distinct malicious nodes.

We have that K is completion-friendly as $\mathcal{P}_{routing}$ and P_0 are both completion-friendly. Moreover, we can deduce, by Lemma 6.1, that there exist two non-neighboring nodes

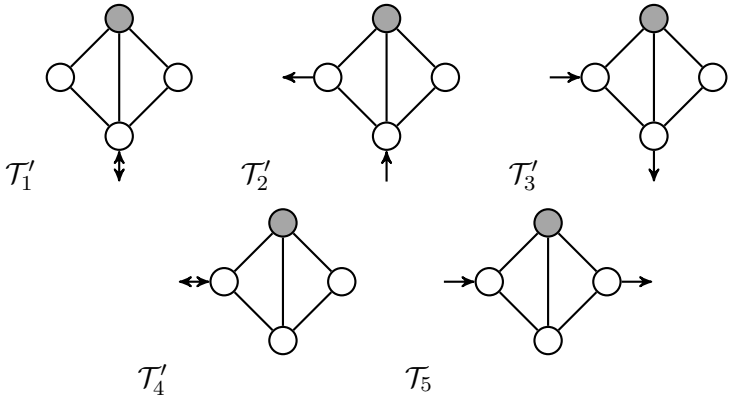
$B, C \in V$ that are not both malicious and a topology $\mathcal{T}^+ = (G^+, \mathcal{M}, S, D)$, a quasi-completion of $\mathcal{T}$ with respect to (B, C), such that there is an attack on K in $\mathcal{T}^+$. As $\mathcal{T}^+$ is a quasi-completion of $\mathcal{T}$ with respect to a pair (B, C), then $N_{G^+}(B) = V \setminus \{C\}$, $N_{G^+}(C) = V \setminus \{B\}$, and $N_{G^+}(W) = V$ for any $W \in V \setminus \{B, C\}$. Since we have assumed that V contains at least three distinct nodes that are not in $\mathcal{M}$ and two distinct nodes in $\mathcal{M}$, we deduce that $V \setminus \{B, C\}$ contains at least an honest node let us say A and a malicious one let us say M. Let ρ be a renaming on the agent names such that for any $W \in V \setminus \{B, C\}$, $\rho(W) = A$ if $W \notin \mathcal{M}$ and $\rho(W) = M$ otherwise. We have that ρ preserves honesty and neighborhood. Then, by Lemma 6.2, we deduce that there is an attack on $K' = K\rho$ in $\mathcal{T}' = (G^+\rho, \mathcal{M}\rho, S\rho, D\rho) = \mathcal{T}^+\rho$. The topology $\mathcal{T}'$ has four nodes: one honest, one malicious and two nodes B, C. We distinguish cases depending in the nature of the nodes B and C, which are not both malicious:

1. B honest and C malicious (the reverse is the same due to symmetry). In this case $\mathcal{T}'$ has two honest nodes, thus according to the position of the source and destination we have the following four possibilities:



   Note that the topology $\mathcal{T}_4$ can obtained only if the source and destination are the same in the original topology.

2. Both are honest. So $\mathcal{T}'$ has three honest nodes in this case. Depending on the position of the source and destination we have nine possibilities, but due to symmetry four of them can be eliminated. This results in only five topologies:



   The topologies $\mathcal{T}'_1, \mathcal{T}'_2, \mathcal{T}'_3$ and $\mathcal{T}'_4$ are respectively subsumed by $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ and $\mathcal{T}_4$ since if there is an attack in $\mathcal{T}'_i$ for $i \in \{1, 2, 3, 4\}$, then this attack can be mounted in $\mathcal{T}_i$ where an honest node is now malicious.

So $\mathcal{T}'$ is one of the five topologies $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4$ and $\mathcal{T}_5$. Finally, since $\mathcal{P}_{routing}$ and P_0 do not contain any names, Definition 6.4 is satisfied and thus $K' = (\mathcal{P}\rho, \mathcal{I}\rho)$ is a valid configuration with respect to $\mathcal{T}'$. Hence, we can conclude. □

6.5 Equivalence Result

In this section we show that a given protocol satisfies route validity for every topology in presence of cooperative attackers, if and only if, it satisfies route validity for every topology in presence of non-cooperative (independent) attackers. We note that, Theorem 6.2 allow us to dispense of Theorem 6.1 since, as a consequence of the former, one could use the result of Cortier *et al.* [CDD12] when looking to verify a protocol for every topology. Still Theorem 6.1 shows that one will obtain one of the five presented topologies when apply the reduction procedure to a certain topology.

Lemma 6.3. (Preservation of the attack). *Let K_0 be a configuration that is completion-friendly. If there exists a topology $\mathcal{T}$ such that K_0 admits an attack in $\mathcal{T}$ in presence of cooperative attackers, then there exists a topology $\mathcal{T}'$ that can be obtained from $\mathcal{T}$ such that K_0 admits an attack in $\mathcal{T}'$ in presence of non-cooperative attackers.*

Proof. Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$ and K_0 be a configuration that is completion-friendly. Suppose that there is an attack on K_0 in $\mathcal{T}$ then, by the definition of the attack, there exist A, P, $\mathcal{P}$, $\mathcal{I}$ and $l_0 = [A_1, \ldots, A_n]$ such that $K \rightarrow^*_{\mathcal{T}} (\lfloor out(end(l_0)).P \rfloor_A \cup \mathcal{P}; \mathcal{I})$ and l_0 is an invalid route in $\mathcal{T}$, *i.e.*, there exists $1 \leq a \leq n$ such that $(A_a, A_{a+1}) \notin E$ and ($A_a \notin \mathcal{M}$ or $A_{a+1} \notin \mathcal{M}$). Let $\mathcal{T}' = (G', \mathcal{M}, S, D)$ be a topology such that $G' = (V \cup \{M_0\}, E')$ with $E' = E \cup (\mathcal{M} \cup \{M_0\} \times \mathcal{M} \cup \{M_0\})$, and $M_0 \in \mathcal{M}$ is a special malicious node that is ready to input an infinite number of messages of any form, *i.e.*, $\lfloor P \rfloor_{M_0} = \lfloor P' | !in(w) \rfloor_{M_0}$ with $w \in \mathcal{X} \setminus (vars(K) \cup vars(P'))$. Since l_0 is invalid in $\mathcal{T}$, it is also invalid in $\mathcal{T}'$ according to the definition. To deduce that there is an attack on K in $\mathcal{T}'$ we show that $K \rightarrow^*_{\mathcal{T}'} (\lfloor out(end(l_0)).P \rfloor_A \cup \mathcal{P}; \mathcal{I})$. First, in order to homogenize the initial knowledge of the attackers, the IN rule has to be applied by each malicious node M_i toward node M_0 a certain number of times (equal to the cardinality of its initial knowledge I_i) to transmit its knowledge to other malicious nodes. This is possible since M_0 is ready to input any number of (any) messages. When a node M_i sent a message m to M_0 all malicious nodes will get m too since they are all connected in $\mathcal{T}'$.

Then, we show by induction on the length r of a derivation $K_0 \rightarrow^r_{\mathcal{T}} K_r$ that K_r is completion-friendly and $K_0 \rightarrow^*_{\mathcal{T}'} K_r$. This will allow us to deduce that $K_0 \rightarrow^*_{\mathcal{T}'} (\lfloor out(end(l_0)).P \rfloor_A \cup \mathcal{P}; \mathcal{I})$. For $r = 0$, as $K_r = K_0$, K_0 is completion-friendly, $E \subseteq E'$, and ($A_a \notin \mathcal{M}$ or $A_{a+1} \notin \mathcal{M}$), then we can easily conclude. Assume that $K_0 \rightarrow^{r-1}_{\mathcal{T}'} K_{r-1}$ then, by induction hypothesis, we have that K_{r-1} is completion-friendly and $K_0 \rightarrow^*_{\mathcal{T}'} K_{r-1}$.

Suppose that, $K_{r-1} \rightarrow_{\mathcal{T}} K_r$. Since K_{r-1} is completion-friendly, then K_r is completion-friendly as there is no rule that introduce new predicates or functions. We show that, by case analysis on the rule involved in the step $K_{r-1} \rightarrow_{\mathcal{T}} K_r$ with cooperative attackers, the equivalence rule or rules that can be applied in $\mathcal{T}'$ with non-cooperative attackers to get $K_{r-1} \rightarrow^*_{\mathcal{T}'} K_r$:

- **Case of the rule IF-THEN:** since K is completion friendly then any formula ϕ that is true for $\mathcal{T}$, its true also for $\mathcal{T}'$. Thus, the rule IF-THEN can also be applied in $\mathcal{T}'$ in this case.
- **Case of the rule IN:** since $E \subseteq E'$ the same rule can be applied and as all malicious nodes are connected in $\mathcal{T}'$ the sent message is received by all attackers so the knowledge remains equal.
- **Case of the COMM:** Since $E \subseteq E'$ we can apply the rule COMM. However, in case where we have a malicious node M_i that is a neighbour of the sender of the message, then the rule COMM should be followed by an application of rule IN from M_i toward M_0. So that, all other malicious nodes will get that message. This last step is for a malicious node that has received a message to share it in an indirect way with all the other malicious nodes.
- **Case of the rules PAR, REPL, and NEW:** these rules do not depend on the underlying graph. So same rules can be applied in $\mathcal{T}'$.

□

Theorem 6.2 (Equivalence)**.** *Let $\mathcal{P}_{routing}$ be a routing protocol and P_0 be a routing role. Then, $\mathcal{P}_{routing}$ and P_0 are secure for every topology in presence of cooperative attackers with a knowledge $\mathcal{I}$, if and only if, $\mathcal{P}_{routing}$ and P_0 are secure for every topology in presence of non-cooperative attackers, each with the same knowledge $\mathcal{I}$.*

Proof. First direction: the non-cooperative attackers have weaker abilities than the cooperative ones. So, if there is no attack on $\mathcal{P}_{routing}$ and P_0 given $\mathcal{I}$ in presence of cooperative attackers then there is no attack on $\mathcal{P}_{routing}$ and P_0 in presence of non-cooperative attackers.

Second direction: Suppose that there is no attack on $\mathcal{P}_{routing}$ and P_0 given $\mathcal{I}$ for any topology with non-cooperative attackers. Assume that there exists a topology $\mathcal{T}$ such that there is an attack on $\mathcal{P}_{routing}$ and P_0 given $\mathcal{I}$ in $\mathcal{T}$ with cooperative attackers. Then, by the definition of the attack 6.5, there is a configuration K that is valid for $\mathcal{P}_{routing}$ and P_0, such that there is an attack on K in $\mathcal{T}$ with cooperative attackers. Thus, by Lemma 6.3, there is a topology $\mathcal{T}'$ that can be obtained from $\mathcal{T}$ such that there is an attack in it on the configuration K with non-cooperative attackers. Hence, there is an attack on $\mathcal{P}_{routing}$ and P_0 given $\mathcal{I}$ in $\mathcal{T}'$ which leads to a contradiction. □

Note that, by considering one fixed topology $\mathcal{T}$ this equivalence does not hold anymore as we can find an attack on a protocol in $\mathcal{T}$ in presence of cooperative attackers, while this protocol is secure in $\mathcal{T}$ in presence of non-cooperative attackers. This due to the fact that with cooperative attackers, we give the malicious nodes a powerful capability (*i.e.*, sharing of knowledge) that is difficult to exist in reality, which may result into attacks that can not be mounted in practice. Having a fixed known topology one could choose in presence of which kind of the attacker to verify the used protocol, depending on the application and the level of the security required.

6.6 Conclusion

We consider multiple independent attackers (compromised nodes) that do not have the ability to share their knowledge with each other. We extend the reduction proof proposed in [CDD12] to the case of multiple independent attackers. That is when looking for attacks on route validity, it is sufficient to verify only five topologies with four nodes each. This entails updating the transition execution rules to handle the case of independent attackers, and revisiting the proofs. We also show that a protocol is secure in any topology with cooperative attackers, if and only if, it is secure for any topology with independent (non-cooperative) attackers.

For future work, we would like to conduct some case studies for analyzing route validity in presence of independent attackers. It could be interesting to develop an automatic tool that is able to analyze route validity in presence of multiple independent attackers. Another research direction is to study route validity in wireless sensor networks with attackers that only have a limited power for communications (a battery).

Chapter 7

Conclusion

In this document we provide symbolic definitions of security properties for exam, e-cash, and e-reputation protocols. Particularly, we studied authentication, privacy and verifiability properties of these protocols. Moreover, we develop a reduction result for verification of route validity in presence of multiple independent attackers. We also provide a case study concerning runtime verification of e-exams. In this chapter, we summarize our results, and we discuss the limitations of our work and directions for future research.

7.1 Summary

In Chapter 1, we introduced the context and motivated our work. We recalled the syntax and semantics of the Applied π-Calculus in Chapter 2. We also provided a brief overview of ProVerif tool, QEA syntax, and MarQ tool.

We then studied the security of exam protocols in Chapter 3. We formalize nine authentication and privacy properties in the Applied π-Calculus, and automatically analyzed using ProVerif three case studies: the protocols due to Huszti & Pethő [HP10], Giustolisi *et al.* [GLR14], and *Université Grenoble Alpes*[1] exam (Grenoble exam). Our analysis shows that Huszti & Pethő protocol satisfies none of the nine properties. Authentication is compromised because of inaccuracies in the protocol design, whereas most of attacks invalidating privacy exploit a vulnerability in a component that the protocol uses, namely the RARC. The attacks compromise secrecy and anonymity of the messages taking advantage of the absence of a proof of knowledge of the submitted message to the RARC, a vulnerability that allows the attacker to use the RARC as a decryption oracle. We proposed a few modifications on the H&P protocol in order to guarantee a subset of the authentication properties. ProVerif confirms that the modified protocol ensures these properties. However, even when assuming a perfect RARC ensuring anonymity,

[1] www.univ-grenoble-alpes.fr

we still have attacks on privacy properties. Thus, we think that fixing the RARC is not sufficient, and that the protocol requires fundamental changes. Remark! protocol presents a weakness that violates *Form Authenticity* when a corrupted candidate is considered. We propose a fix and formally verify that the fixed protocol satisfies all the properties herein considered. Grenoble exam satisfies seven properties (all but *Anonymous Examiner* and *Mark Privacy*). ProVerif finds a counterexample against *Anonymous Examiner* that is difficult to mount in practice. The attacker can distinguish which "unseen"[2] exam-form the examiner accepts for marking (the one he can "seen" using his secret key). This is not a real attack, since the examiner will only accept exam-form from the exam authority, not an attacker. Assuming a private channel between the exam authority and the examiner, ProVerif confirms that *Anonymous Examiner* is satisfied by Grenoble exam. Concerning *Mark Privacy*, ProVerif finds an attack (when all parties are honest), this was expected as in Grenoble exam the marks are published in clear-text by the exam authority.

In the second part of the Chapter 3, we abstractly defined exam Verifiability. We identified eleven verifiability properties. For each property, we define the soundness and completeness conditions for the related verification test. Then, we analyzed with the help of ProVerif two exam protocols: the protocol due to Giustolisi *et al.* [GLR14], and Grenoble exam. The analysis of Giustolisi *et al.* shows that all properties but three are satisfied without assuming that the exam's roles are honest. But *Marking Correctness* holds only assuming an honest exam authority. In fact, a student can check her mark by using the exam table, but this is posted on the bulletin board by the exam authority who can nullify the verification of correctness by tampering with the table. Whereas the analysis of Grenoble exam shows that it satisfies all the verifiability properties under the assumption that authorities and examiner are honest. This seems to be peculiar to paper-and-pencil exams, where log-books and registers are managed by the authorities that can tamper with them. Only *Marking Correctness* holds even in presence of dishonest authorities and examiner: here, a candidate can consult her exam-test after marking, thus verifying herself whether her mark has been computed correctly.

In the final part of the Chapter 3, we defined monitors that allows to check exam requirements at runtime. Then, we implemented these monitors and analyzed logs extracted from real e-exam organized by *Université Joseph Fourier* (UJF) using MarQ tool. The analysis reveals some students that violate the requirements, and also some discrepancies between the specification and the implementation. Due to the lack of logs about the marking and notification phases, we were not able to analyze all properties. The UJF exam case study clearly demonstrates that the developers do not think to log these two phases where there is less interaction with the candidates. However, we believe that monitoring the marking phase is essential since a successful attempt from a bribed examiner or a cheating student can be very effective.

[2] `unseen` function is similar to an asymmetric encryption.

Next in Chapter 4, we define security properties of e-cash protocols We proposed two client privacy properties and three properties to prevent forgery. Then, we applied our definitions using ProVerif to analyze the Chaum protocol [Cha82], the DigiCash protocol presented in [Sch97], and the Chaum *et al.* protocol [CFN88]. Our analysis confirms several already known attacks, and reveals a flaw that confirms the necessity of synchronization in online cash protocols.

In Chapter 5, we established a formal framework for the security analysis of e-reputation protocols. In particular, we show how to model reputation protocols in the Applied π-Calculus, and defined eight privacy, authentication, and verifiability properties. We validate our definitions by analyzing, using ProVerif, the security of a simple e-reputation protocol, the protocol by Pavlov *et al.* [PRT04]. It has been informally argued to preserve rate privacy. Our analysis shows that it satisfies five properties, namely *Rate Privacy*, *Rate Anonymity*, and *User Eligibility*, *Rate Integrity* (without injectivity), and *Reputation Score Verification* (if users can have an access to the set of rates). While it fails to satisfy the other five: *User Privacy*, *Untraceability*, *Receipt-Freeness*, *Coercion-Resistance*, and *User Eligibility Verification*.

Finally in Chapter 6, we considered the non-cooperative attacker model where there are multiple attackers working independently, so that no one shares any of its knowledge with the others. We then extended the reduction proof proposed in [CDD12] to the case of multiple independent attackers. That is when looking for attacks on route validity, it is sufficient to verify only five topologies with four nodes each. This entails updating the transition execution rules to handle the case of independent attackers, and revisiting the required proofs. We also showed that a protocol is secure in any topology with cooperative attackers, if and only if, it is secure for any topology with independent attackers.

7.2 Limitations and Directions for Future Works

Concerning the work on exam protocols several avenues for future works are open. The current model of the protocol by Huszti & Pethő used for privacy an authentication properties is a simplification, and we did not model the full protocol with RARC due to ProVerif termination problem (instead we analyzed RARC alone). It would be interesting to extend this to a more precise model. The main obstacles are too complex equational theories. A step in this direction was undertaken by Smyth *et al.* [SAR13]. They propose to replace the complex equational theory with a simpler, but equivalent one which can then be treated by ProVerif. Moreover, KISS [cCDK12] and AKISS [CcCK12] can deal with more complex equational theories.

Concerning verifiability, we use ProVerif to verify universal verifiability properties, we were unable to prove the general case directly and had to provide a manual proof to

show that the result holds for any number of candidates. It would be great to automatize these proofs. Note that, an extension of ProVerif with loops could permit to obtain the general proof directly. Another line of research is to study novel properties such as accountability, which allows them to identify which party is responsible when the verification fails. Few works go in this direction: Küsters *et al.* [KTV10] have studied accountability, however, their framework needs to be instantiated for each application by identifying relevant verifiability goals. As we identified several verifiability properties relevant for exam protocols, one may study how their accountability framework can be applied to case of exam protocols. Bella *et al.* [BGLR15] have proposed an accountability property, which allows to identify the responsible party when a candidate fails to submit an answer or to receive the corresponding mark. They have analyzed their protocol and have shown that it satisfies this property. However, more accountability properties need to be defined to identify the responsible parties when the verifiability properties we propose in this thesis fails.

With respect to runtime verification, one direction is to perform the verification of marking related properties, as well as, to perform online verification with our monitors during live e-exams, and to study to what extent runtime enforcement can be applied during a live e-exam run. Online verification requires to deliver events to the monitor at the time they are generated by the system in order to check them and take a verdict. This introduces additional overheads and may cause scalability problems depending on the size of the system and number of events generated *per* second. Another direction is to study more expressive and quantitative properties that might detect possible collusion between students through similar answer patterns.

For e-cash protocols, an interesting line of research is to extend our model to cover transferable protocols with divisible coins. A limitation in our analysis of Chaum *et al.* protocol is that we did not model the details of “cut-and-choose" at withdrawal phase. A computational variant of our definition would be able to take this into account. Actually in symbolic model we did not consider probabilities. So even if there is a low probability for the attacker to guess the parameters the bank requests, then such possibility is considered. With respect to the flaw we found on online cash protocols is due to lack of synchronization at deposit. We did not verify a fixed version that make synchronization. This is due to a limitation of ProVerif when dealing with tables. To confirm our argument concerning the need for synchronization, one could use the tool SAPIC based on Tamarin. SAPIC is suitable to the protocols that entail database lookups. It offers the ability to lock a table then to unlock it after the end of operations, a feature that can be used to model synchronization.

For reputation protocols, one direction is to develop a tool that can deal with algebraic properties such as arithmetic operations and homomorphic encryptions as e-reputation

protocols are highly depends on such algebraic properties. Note that, developing such a tool is still a real challenge for the community. However, researches try to find solutions for such a problem. For instance the result obtained by Küsters *et al.* [KT11] allows us to analyze protocols with XOR operator for trace based properties using ProVerif. Extending this result to handle equivalence properties could help in analyzing protocols with arithmetic operations, which is the case of many e-reputation protocols, if XOR operator is used to model summation and subtraction. With respect to privacy properties, we discussed the *Receipt-Freeness* and *Coercion-Resistance* only informally on our case study. It would be great to construct a formal proofs for them. Other interesting research works include the study of the relation between our security properties as well as the definition of novel properties such as correctness of the reputation score, preventing double rating, accountability, reliability (*e.g.*, to prevent Sybil attacks), and addressing fake rates.

Research directions for wireless sensor networks include performing some case studies for analyzing route validity in presence of independent attackers. It could be interesting to develop an automatic tool that is able to analyze route validity in presence of multiple independent attackers Another research direction is to study route validity in wireless sensor networks with attackers (compromised nodes) that only have a limited power for communications (a battery).

Bibliography

[AA14] Sattar J. Aboud and Ammar Agoun. Article: Analysis of a known offline e-coin system. *International Journal of Computer Applications*, 98(15):27–30, July 2014. Full text available.

[AAA+12] Alessandro Armando, Wihem Arsac, Tigran Avanesov, Michele Barletta, Alberto Calvi, Alessandro Cappai, Roberto Carbone, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Gabriel Erzse, Simone Frau, Marius Minea, Sebastian Mödersheim, David von Oheimb, Giancarlo Pellegrino, Serena Elisa Ponta, Marco Rocchetto, Michaël Rusinowitch, Mohammad Torabi Dashti, Mathieu Turuani, and Luca Viganò. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7214 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2012.

[AAC+05] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie J. Hendren, Sascha Kuzins, Ondrej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to aspectj. In Ralph E. Johnson and Richard P. Gabriel, editors, *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, October 16-20, 2005, San Diego, CA, USA*, pages 345–364. ACM, 2005.

[AB05a] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *J. ACM*, 52(1):102–146, 2005.

[AB05b] Martín Abadi and Bruno Blanchet. Computer-assisted verification of a protocol for certified email. *Sci. Comput. Program.*, 58(1-2):3–27, 2005.

[AB05c] Xavier Allamigeon and Bruno Blanchet. Reconstruction of attacks against cryptographic protocols. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005), 20-22 June 2005, Aix-en-Provence, France*, pages 140–154. IEEE Computer Society, 2005.

[ABB+05] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[ABF07] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just fast keying in the pi calculus. *ACM Trans. Inf. Syst. Secur.*, 10(3), 2007.

[ABR13] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Privacy-supporting cloud computing by in-browser key translation. *Journal of Computer Security*, 21(6):847–880, 2013.

[ÁBV06] Gergely Ács, Levente Buttyán, and István Vajda. Provably secure on-demand source routing in mobile ad hoc networks. *IEEE Trans. Mob. Comput.*, 5(11):1533–1546, 2006.

[ABY11] Todd R. Andel, G. Back, and Alec Yasinsac. Automating the security analysis process of secure ad hoc routing protocols. *Simulation Modelling Practice and Theory*, 19(9):2032–2049, 2011.

[ACBM08] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. Reputation systems for anonymous networks. In Nikita Borisov and Ian Goldberg, editors, *Privacy Enhancing Technologies*, volume 5134 of *Lecture Notes in Computer Science*, pages 202–218. Springer, 2008.

[ACC+08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, and M. Llanos Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. In Vitaly Shmatikov, editor, *Proceedings of the 6th ACM Workshop on Formal*

Methods in Security Engineering, FMSE 2008, Alexandria, VA, USA, October 27, 2008, pages 1–10. ACM, 2008.

[ACC14] Alessandro Armando, Roberto Carbone, and Luca Compagna. SATMC: A sat-based model checker for security-critical systems. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2014.

[ACD10] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. Modeling and verifying ad hoc routing protocols. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 59–74. IEEE Computer Society, 2010.

[ACRT11] Tigran Avanesov, Yannick Chevalier, Michaël Rusinowitch, and Mathieu Turuani. Satisfiability of general intruder constraints with and without a set constructor. *CoRR*, abs/1103.0220, 2011.

[AD01] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA, November 5-10, 2001*, pages 310–317. ACM, 2001.

[ADMPQ09] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/-WOTE'09, pages 10–10, Berkeley, CA, USA, 2009. USENIX Association.

[AF96] Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, volume 1163 of *Lecture Notes in Computer Science*, pages 244–251. Springer, 1996.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on*

Principles of Programming Languages, London, UK, January 17-19, 2001, pages 104–115. ACM, 2001.

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In Richard Graveman, Philippe A. Janson, Clifford Neumann, and Li Gong, editors, *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997.*, pages 36–47. ACM, 1997.

[AGLM+13a] Emmanuelle Anceaume, Gilles Guette, Paul Lajoie-Mazenc, Nicolas Prigent, and Valérie Viet Triem Tong. A privacy preserving distributed reputation mechanism. In *ICC*, pages 1951–1956. IEEE, 2013.

[AGLM+13b] Emmanuelle Anceaume, Gilles Guette, Paul Lajoie-Mazenc, Thomas Sirvent, and Valérie Viet Triem Tong. Extending signatures of reputation. In Marit Hansen, Jaap-Henk Hoepman, Ronald E. Leenes, and Diane Whitehouse, editors, *Privacy and Identity Management*, volume 421 of *IFIP Advances in Information and Communication Technology*, pages 165–176. Springer, 2013.

[AH91] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1991.

[AH00] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *33rd Annual Hawaii International Conference on System Sciences (HICSS-33), 4-7 January, 2000, Maui, Hawaii, USA*. IEEE Computer Society, 2000.

[Ake70] George A. Akerlof. The market for "lemons": Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics*, 84(3):488–500, 1970.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 75(1):3–51, 2008.

[BAN90] Michael Burrows, Martín Abadi, and Roger M. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[BBF+11] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. *ACM Trans. Program. Lang. Syst.*, 33(2):8, 2011.

[BBF14] Ezio Bartocci, Borzoo Bonakdarpour, and Yliès Falcone. First international competition on software for runtime verification. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, volume 8734 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2014.

[BCCR11] Giampaolo Bella, Gianpiero Costantino, Lizzie Coles-Kemp, and Salvatore Riccobene. Remote management of face-to-face written authenticated though anonymous exams. In Alexander Verbraeck, Markus Helfert, José Cordeiro, and Boris Shishkov, editors, *CSEDU 2011 - Proceedings of the 3rd International Conference on Computer Supported Education, Volume 2, Noordwijkerhout, Netherlands, 6-8 May, 2011*, pages 431–437. SciTePress, 2011.

[BCE+14] David A. Basin, Germano Caronni, Sarah Ereth, Matús Harvan, Felix Klaedtke, and Heiko Mantel. Scalable offline monitoring. In *RV 2014*, pages 31–47, 2014.

[BCR10] Giampaolo Bella, Gianpiero Costantino, and Salvatore Riccobene. WATA - A system for written authenticated though anonymous exams. In José A. Moinhos Cordeiro, Boris Shishkov, Alexander Verbraeck, and Markus Helfert, editors, *CSEDU 2010 - Proceedings of the Second International Conference on Computer Supported Education, Valencia, Spain, April 7-10, 2010 - Volume 2*, pages 132–137. INSTICC Press, 2010.

[Ben96] Josh Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, Dec. 1996.

[BFG10] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Modular verification of security protocol code by typing. In Manuel V. Hermenegildo and Jens Palsberg, editors, *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 445–456. ACM, 2010.

[BFGT08] Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Stephen Tse. Verified interoperable implementations of security protocols. *ACM Trans. Program. Lang. Syst.*, 31(1), 2008.

[BFH+12] Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In *FM 2012*, 2012.

[BGHS04] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, January 11-13, 2004, Proceedings*, volume 2937 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2004.

[BGL14] Giampaolo Bella, Rosario Giustolisi, and Gabriele Lenzini. Secure exams despite malicious management. In Ali Miri, Urs Hengartner, Nen-Fu Huang, Audun Jøsang, and Joaquín García-Alfaro, editors, *2014 Twelfth Annual International Conference on Privacy, Security and Trust, Toronto, ON, Canada, July 23-24, 2014*, pages 274–281. IEEE, 2014.

[BGLR15] Giampaolo Bella, Rosario Giustolisi, Gabriele Lenzini, and Peter Y. A. Ryan. A secure exam protocol without trusted parties. In Hannes Federrath and Dieter Gollmann, editors, *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 495–509. Springer, 2015.

[BH11] Howard Barringer and Klaus Havelund. Tracecontract: A scala DSL for trace analysis. In Michael J. Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, volume 6664 of *Lecture Notes in Computer Science*, pages 57–72. Springer, 2011.

[BHKO04] Yohan Boichut, Pierre-Cyrille Héam, Olga Kouchnarenko, and Frédéric Oehl. Improvements on the Genet and Klay technique to automatically verify security protocols. In *Proc. Int. Ws. on Automated Verification of Infinite-State Systems (AVIS'2004), joint to ETAPS'04*, pages 1–11, Barcelona, Spain, apr 2004.

[BHL+10] Eric Bodden, Laurie J. Hendren, Patrick Lam, Ondrej Lhoták, and Nomair A. Naeem. Collaborative runtime verification with tracematches. *J. Log. Comput.*, 20(3):707–723, 2010.

[BHM08] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, 23-25 June 2008*, pages 195–209. IEEE Computer Society, 2008.

[BHRG09] Howard Barringer, Klaus Havelund, David E. Rydeheard, and Alex Groce. Rule systems for runtime verification: A short tutorial. In Saddek Bensalem and Doron Peled, editors, *Runtime Verification, 9th International Workshop, RV 2009, Grenoble, France, June 26-28, 2009. Selected Papers*, volume 5779 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2009.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96. IEEE Computer Society, 2001.

[Bla02] Bruno Blanchet. From secrecy to authenticity in security protocols. In Manuel V. Hermenegildo and Germán Puebla, editors, *Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17-20, 2002, Proceedings*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359. Springer, 2002.

[Bla04] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, page 86. IEEE Computer Society, 2004.

[BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 202–215. IEEE Computer Society, 2008.

[BMV05] David A. Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.

[BMV10] Davide Benetti, Massimo Merro, and Luca Viganò. Model checking ad hoc network routing protocols: ARAN vs. endaira. In José Luiz Fiadeiro, Stefania Gnesi, and Andrea Maggiolo-Schettini, editors, *8th IEEE International Conference on Software Engineering and Formal Methods, SEFM*

2010, Pisa, Italy, 13-18 September 2010, pages 191–202. IEEE Computer Society, 2010.

[Bra93] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 1993.

[Bra06] Felix Brandt. How to obtain full privacy in auctions. *Int. J. Inf. Sec.*, 5(4):201–216, 2006.

[BRH10] Howard Barringer, David E. Rydeheard, and Klaus Havelund. Rule systems for run-time monitoring: from eagle to ruler. *J. Log. Comput.*, 20(3):675–706, 2010.

[BSC15] Bruno Blanchet, Ben Smyth, and Vincent Cheval. *ProVerif 1.90: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2015. Originally appeared as Bruno Blanchet and Ben Smyth (2011) ProVerif 1.85: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.

[BSS10] John Bethencourt, Elaine Shi, and Dawn Song. Signatures of reputation. In Radu Sion, editor, *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 400–407. Springer, 2010.

[BT94] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC '94*, pages 544–553, New York, NY, USA, 1994. ACM.

[BV04] Levente Buttyán and István Vajda. Towards provable security for ad hoc routing protocols. In Sanjeev Setia and Vipin Swarup, editors, *Proceedings of the 2nd ACM Workshop on Security of ad hoc and Sensor Networks, SASN 2004, Washington, DC, USA, October 25, 2004*, pages 94–105. ACM, 2004.

[CcCK12] Rohit Chadha, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2012.

[cCDK12] Ştefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. *J. Autom. Reasoning*, 48(2):219–262, 2012.

[CD09] Sagar Chaki and Anupam Datta. ASPIER: an automated framework for verifying security protocol implementations. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, pages 172–185. IEEE Computer Society, 2009.

[CDD12] Véronique Cortier, Jan Degrieck, and Stéphanie Delaune. Analysing routing protocols: Four nodes topologies are sufficient. In Pierpaolo Degano and Joshua D. Guttman, editors, *POST*, volume 7215 of *Lecture Notes in Computer Science*, pages 30–50. Springer, 2012.

[CF85] Josh Cohen and Michael Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *FOCS'85*, pages 372–382. IEEE Computer Society, October 1985.

[CFN88] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer, 1988.

[CG08] Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, New York, NY, USA, June 3-6, 2008. Proceedings*, volume 5037 of *Lecture Notes in Computer Science*, pages 207–223, 2008.

[CGT08] Sébastien Canard, Aline Gouget, and Jacques Traoré. Improvement of efficiency in (unconditional) anonymous transferable e-cash. In Gene Tsudik, editor, *Financial Cryptography and Data Security, 12th International Conference, FC 2008, Cozumel, Mexico, January 28-31, 2008, Revised Selected Papers*, volume 5143 of *Lecture Notes in Computer Science*, pages 202–214. Springer, 2008.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982.*, pages 199–203. Plenum Press, New York, 1982.

[CHD06] Jordi Castellà-Roca, Jordi Herrera-Joancomartí, and Aleix Dorca-Josa. A secure e-exam management system. In *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES 2006, The International Dependability Conference - Bridging Theory and Practice, April 20-22 2006, Vienna University of Technology, Austria*, pages 864–871. IEEE Computer Society, 2006.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[CK10] Radu Calinescu and Shinji Kikuchi. Formal methods @ runtime. In Radu Calinescu and Ethan K. Jackson, editors, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems - 16th Monterey Workshop 2010, Redmond, WA, USA, March 31- April 2, 2010, Revised Selected Papers*, volume 6662 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2010.

[CK14] Véronique Cortier and Steve Kremer. Formal models and techniques for analyzing security protocols - a tutorial. Technical report, 2014.

[CKK11] Radu Calinescu, Shinji Kikuchi, and Marta Kwiatkowska. Formal methods for the development and verification of autonomic it systems. *Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification*, pages 1–37, 2011.

[CL07] Edmund M. Clarke and Flavio Lerda. Model checking: Software and beyond. *J. UCS*, 13(5):639–649, 2007.

[CLN09] Cas J. F. Cremers, Pascal Lafourcade, and Philippe Nadeau. Comparing state spaces in automatic security protocol analysis. In Véronique Cortier, Claude Kirchner, Mitsuhiro Okada, and Hideki Sakurada, editors, *Formal to Practical Security - Papers Issued from the 2005-2008 French-Japanese Collaboration*, volume 5458 of *Lecture Notes in Computer Science*, pages 70–94. Springer, 2009.

[CP12] Christian Colombo and Gordon J. Pace. Fast-forward runtime monitoring - an industrial case study. In *RV 2012*, pages 214–228, 2012.

[CPS07] Brian Curtis, Josef Pieprzyk, and Jan Seruga. An efficient eauction protocol. In *Proceedings of the The Second International Conference on Availability, Reliability and Security, ARES 2007, The International Dependability Conference - Bridging Theory and Practice, April 10-13 2007, Vienna, Austria*, pages 417–421. IEEE Computer Society, 2007.

[CPS09] Christian Colombo, Gordon J. Pace, and Gerardo Schneider. LARVA — safer monitoring of real-time java programs (tool paper). In Dang Van Hung and Padmanabhan Krishnan, editors, *Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM 2009, Hanoi, Vietnam, 23-27 November 2009*, pages 33–37. IEEE Computer Society, 2009.

[CR09] Feng Chen and Grigore Rosu. Parametric trace slicing and monitoring. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2009.

[Cre94] Giovanni Di Crescenzo. A non-iterative electronic cash system. In Maurizio A. Bonuccelli, Pierluigi Crescenzi, and Rossella Petreschi, editors, *Algorithms and Complexity, Second Italian Conference, CIAC '94, Rome, Italy, February 23-25, 1994, Proceedings*, volume 778 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 1994.

[Cre08a] Cas J. F. Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

[Cre08b] Cas J. F. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 119–128. ACM, 2008.

[CS11] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *Proceedings of the 24th IEEE Computer Security*

Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011, pages 297–311. IEEE Computer Society, 2011.

[CS13] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

[CYS05] Chang Yu Cheng, Jasmy Yunus, and Kamaruzzaman Seman. Estimations on the security aspect of brand's electronic cash scheme. In *19th International Conference on Advanced Information Networking and Applications (AINA 2005), 28-30 March 2005, Taipei, Taiwan*, pages 131–134. IEEE Computer Society, 2005.

[Dam88] Ivan Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 328–335. Springer, 1988.

[DC94] Stefano D'Amiano and Giovanni Di Crescenzo. Methodology for digital money based on general cryptographic tools. In Santis [San95], pages 156–170.

[Del00] Chrysanthos Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *EC*, pages 150–157, 2000.

[DGK+14] Jannik Dreier, Rosario Giustolisi, Ali Kassem, Pascal Lafourcade, Gabriele Lenzini, and Peter Y. A. Ryan. Formal analysis of electronic exams. In Mohammad S. Obaidat, Andreas Holzinger, and Pierangela Samarati, editors, *SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, 28-30 August, 2014*, 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 Javier Lopez and Yongdong Wu, editors, *Information Security Practice and Experience - 11th International Conference, ISPEC 2015, Beijing, China, May 5-8, 2015. Proceedings*, volume 9065 of *Lecture Notes in Computer Science*, pages 514–529. Springer, 2015.

[DJL13] Jannik Dreier, Hugo Jonker, and Pascal Lafourcade. Defining verifiability in e-auction protocols. In Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng, editors, *ASIACCS*, pages 547–552. ACM, 2013.

[DJL14] Jannik Dreier, Hugo Jonker, and Pascal Lafourcade. Secure auctions without cryptography (extended version). Technical report, ETH-Zürich, 2014.

[DJP10] Naipeng Dong, Hugo Jonker, and Jun Pang. Analysis of a receipt-free auction protocol in the applied pi calculus. In *Proc. 7th Workshop on Formal Aspects in Security and Trust (FAST'10)*, volume 6561 of *LNCS*, pages 223–238. Springer, 2010.

[DJP12] Naipeng Dong, Hugo Jonker, and Jun Pang. Formal analysis of privacy in an ehealth protocol. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, volume 7459 of *Lecture Notes in Computer Science*, pages 325–342. Springer, 2012.

[DKL15] Jannik Dreier, Ali Kassem, and Pascal Lafourcade. Formal analysis of e-cash protocols. In Mohammad S. Obaidat, Pascal Lorenz, and Pierangela Samarati, editors, *SECRYPT 2015 - Proceedings of the 12th International Conference on Security and Cryptography, Colmar, Alsace, France, 20-22 July, 2015.*, pages 65–75. SciTePress, 2015.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying properties of electronic voting protocols. In *Proceedings of the IAVoSS Workshop On Trustworthy Elections (WOTE'06)*, pages 45–52, Cambridge, UK, jun 2006.

[DKR09] Stéphanie 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.

[DKS10] Stéphanie Delaune, Steve Kremer, and Graham Steel. Formal security analysis of pkcs#11 and proprietary extensions. *Journal of Computer Security*, 18(6):1211–1245, 2010.

[DLL11] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. Vote-independence: A powerful privacy notion for voting protocols. In Joaquín García-Alfaro and Pascal Lafourcade, editors, *Foundations and Practice of Security - 4th Canada-France MITACS Workshop, FPS 2011, Paris,*

France, May 12-13, 2011, Revised Selected Papers, volume 6888 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2011.

[DLL12a] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. Defining privacy for weighted votes, single and multi-voter coercion. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, volume 7459 of *Lecture Notes in Computer Science*, pages 451–468. Springer, 2012.

[DLL12b] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. A formal taxonomy of privacy in voting protocols. In *Proceedings of IEEE International Conference on Communications, ICC 2012, Ottawa, ON, Canada, June 10-15, 2012*, pages 6710–6715. IEEE, 2012.

[DLL13] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. Formal verification of e-auction protocols. In David A. Basin and John C. Mitchell, editors, *Principles of Security and Trust - Second International Conference, POST 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7796 of *Lecture Notes in Computer Science*, pages 247–266. Springer, 2013.

[DLM04] Nancy A. Durgin, Patrick Lincoln, and John C. Mitchell. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[Don10] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 167–170. Springer, 2010.

[Dre13] Jannik Dreier. Formal Verification of Voting and Auction Protocols From Privacy to Fairness and Verifiability. PhD thesis, Université de Grenoble, 2013.

[DRS08] Stéphanie Delaune, Mark Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi calculus. In Yücel Karabulut, John Mitchell, Peter Herrmann, and Christian Damsgaard Jensen, editors, *Trust Management II - Proceedings of IFIPTM 2008: Joint iTrust and PST Conferences on Privacy, Trust Management and Security, June 18-20,*

2008, Trondheim, Norway, volume 263 of *IFIP Advances in Information and Communication Technology*, pages 263–278. Springer, 2008.

[Dru00] Doron Drusinsky. The temporal rover and the ATG rover. In Klaus Havelund, John Penix, and Willem Visser, editors, *SPIN Model Checking and Software Verification, 7th International SPIN Workshop, Stanford, CA, USA, August 30 - September 1, 2000, Proceedings*, volume 1885 of *Lecture Notes in Computer Science*, pages 323–330. Springer, 2000.

[DSHJ10] Nitish Dalal, Jenny Shah, Khushboo Hisaria, and Devesh Jinwala. A comparative analysis of tools for verification of security protocols. *IJCNS*, 3(10):779–787, 2010.

[DY81] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 350–357. IEEE Computer Society, 1981.

[DY83a] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[DY83b] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[EGS85] Shimon Even, Oded Goldreich, and Adi Shamir. On the security of ping-pong protocols when implemented using the RSA. In Hugh C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 58–72. Springer, 1985.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Fer93] Niels Ferguson. Single term off-line coins. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 318–328. Springer, 1993.

[FHY13] Chun-I Fan, Vincent Shi-Ming Huang, and Yao-Chun Yu. User efficient recoverable off-line e-cash scheme with fast anonymity revoking. *Mathematical and Computer Modelling*, 58(1-2):227–237, 2013.

[Fig15] Le Figaro. Etudiants: les examens sur tablettes numériques appellés à se multiplier. Press release, January 2015. Available at goo.gl/ahxQJD.

[FJ95] Simon N. Foley and Jeremy Jacob. Specifying security for computer supported collaborative working. *Journal of Computer Security*, 3(4):233–254, 1995.

[Fre78] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, page 215, 1978.

[GFN09] Nataliya Guts, Cédric Fournet, and Francesco Zappa Nardelli. Reliable evidence: Auditability by typing. In *ESORICS'09*, volume 5789 of *LNCS*, pages 168–183. Springer, 2009.

[GJ03] Philippe Golle and Markus Jakobsson. Reusable anonymous return channels. In Sushil Jajodia, Pierangela Samarati, and Paul F. Syverson, editors, *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society, WPES 2003, Washington, DC, USA, October 30, 2003*, pages 94–100. ACM, 2003.

[GLB13] Rosario Giustolisi, Gabriele Lenzini, and Giampaolo Bella. What security for electronic exams? In Bruno Crispo, Ravi S. Sandhu, Nora Cuppens-Boulahia, Mauro Conti, and Jean-Louis Lanet, editors, *2013 International Conference on Risks and Security of Internet and Systems (CRiSIS), La Rochelle, France, October 23-25, 2013*, pages 1–5. IEEE, 2013.

[GLR14] Rosario Giustolisi, Gabriele Lenzini, and Peter Y. A. Ryan. Remark!: A secure protocol for remote exams. In Bruce Christianson, James A. Malcolm, Vashek Matyás, Petr Svenda, Frank Stajano, and Jonathan Anderson, editors, *Security Protocols XXII - 22nd International Workshop Cambridge, UK, March 19-21, 2014 Revised Selected Papers*, volume 8809 of *Lecture Notes in Computer Science*, pages 38–48. Springer, 2014.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377. ACM, 1982.

[Gou05] Jean Goubault-Larrecq. Deciding H_1 by resolution. *Inf. Process. Lett.*, 95(3):401–408, 2005.

[GP05] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic protocol analysis on real C code. In Radhia Cousot, editor, *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings*, volume 3385 of *Lecture Notes in Computer Science*, pages 363–379. Springer, 2005.

[Hav15] Klaus Havelund. Rule-based runtime verification revisited. *STTT*, 17(2):143–170, 2015.

[HBBS13] Omar Hasan, Lionel Brunie, Elisa Bertino, and Ning Shang. A decentralized privacy preserving reputation protocol for the malicious adversarial model. *IEEE Transactions on Information Forensics and Security*, 8(6):949–962, 2013.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HP10] Andrea Huszti and Attila Pethő. A secure electronic exam system. *Publicationes Mathematicae Debrecen*, 77(3-4):299–312, 2010.

[HPJ05] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless Networks*, 11(1-2):21–38, 2005.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT'00*, volume 1807 of *LNCS*, pages 539–556. Springer, 2000.

[HS11] Rolf Haenni and Oliver Spycher. Secure internet voting on limited devices with anonymized dsa public keys. In *EVT/WOTE'11*. USENIX, 2011.

[HSD+05] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 2–15. ACM, 2005.

[HW06] Daniel Houser and John Wooders. Reputation in Auctions: Theory, and Evidence from eBay. *Journal of Economics & Management Strategy*, 15(2):353–369, 06 2006.

[ISO12] ISO 15408-2: Common Criteria for Information Technology Security Evaluation - Part 2: Security functional components. *ISO/IEC*, Version 3.1, Revision 4, September 2012.

[JMB01] David B. Johnson, David A. Maltz, and Josh Broch. In ad hoc networking. chapter DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks, pages 139–172. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[JMLR12] Dongyun Jin, Patrick O'Neil Meredith, Choonghwan Lee, and Grigore Rosu. Javamop: Efficient parametric runtime monitoring framework. In Martin Glinz, Gail C. Murphy, and Mauro Pezzè, editors, *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 1427–1430. IEEE, 2012.

[Jür06] Jan Jürjens. Security analysis of crypto-based java programs using automated theorem provers. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006), 18-22 September 2006, Tokyo, Japan*, pages 167–176. IEEE Computer Society, 2006.

[KFL15] Ali Kassem, Yliès Falcone, and Pascal Lafourcade. Monitoring electronic exams. In *The 15th International Conference on Runtime Verification (RV'15)*, 2015.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In Jørgen Lindskov Knudsen, editor, *ECOOP 2001 - Object-Oriented Programming, 15th European Conference, Budapest, Hungary, June 18-22, 2001, Proceedings*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer, 2001.

[KK14] Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 163–178. IEEE Computer Society, 2014.

[KLL13] Ali Kassem, Pascal Lafourcade, and Yassine Lakhnech. A more realistic model for verifying route validity in ad-hoc networks. In Jean Luc Danger, Mourad Debbabi, Jean-Yves Marion, Joaquín García-Alfaro, and A. Nur Zincir-Heywood, editors, *Foundations and Practice of Security - 6th International Symposium, FPS 2013, La Rochelle, France, October 21-22, 2013, Revised Selected Papers*, volume 8352 of *Lecture Notes in Computer Science*, pages 306–322. Springer, 2013.

[KLL14] Ali Kassem, Pascal Lafourcade, and Yassine Lakhnech. Formal verification of e-reputation protocols. In Frédéric Cuppens, Joaquín García-Alfaro, A. Nur Zincir Heywood, and Philip W. L. Fong, editors, *Foundations and*

Practice of Security - 7th International Symposium, FPS 2014, Montreal, QC, Canada, November 3-5, 2014. Revised Selected Papers, volume 8930 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 2014.

[KO02] Sangjin Kim and Heekuck Oh. A new electronic check system with reusable refunds. *International Journal of Information Security*, 1(3):175–188, 2002.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KR05] Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In Shmuel Sagiv, editor, *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2005.

[KRS10a] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[KRS10b] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *ESORICS'10*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010.

[KT09] Ralf Küsters and Tomasz Truderung. Using proverif to analyze protocols with diffie-hellman exponentiation. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, pages 157–171. IEEE Computer Society, 2009.

[KT11] Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with XOR to the xor-free case in the horn theory based approach. *J. Autom. Reasoning*, 46(3-4):325–352, 2011.

[KTAK07] Shinji Kikuchi, Satoshi Tsuchiya, Motomitsu Adachi, and Tsuneo Katsuyama. Policy verification and validation framework based on model checking approach. In *Fourth International Conference on Autonomic*

Computing (ICAC'07), Jacksonville, Florida, USA, June 11-15, 2007, page 1. IEEE Computer Society, 2007.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535. ACM, 2010.

[KVB+99] Moonjoo Kim, Mahesh Viswanathan, Hanêne Ben-Abdallah, Sampath Kannan, Insup Lee, and Oleg Sokolsky. Formally specified monitoring of temporal properties. In *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*, pages 114–122. IEEE Computer Society, 1999.

[Lar13] Larry Copeland. School cheating scandal shakes up atlanta. USA TODAY, April 2013. Available at http://goo.gl/wGr40s.

[LBaK+98] Insup Lee, Hanêne Ben-abdallah, Sampath Kannan, Moonjoo Kim, Oleg Sokolsky, , and Mahesh Viswanathan. A monitoring and checking framework for run-time correctness assurance. In *Proceedings of the 1998 Korea-U.S. Technical Conference on Strategic Technologies*, 1998.

[LCH+05] Boon Thau Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Implementing declarative overlays. In Andrew Herbert and Kenneth P. Birman, editors, *Proceedings of the 20th ACM Symposium on Operating Systems Principles 2005, SOSP 2005, Brighton, UK, October 23-26, 2005*, pages 75–90. ACM, 2005.

[LCPD07] Zhengqin Luo, Xiaojuan Cai, Jun Pang, and Yuxin Deng. Analyzing an electronic cash protocol using applied pi calculus. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*, volume 4521 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2007.

[LHSR05] Boon Thau Loo, Joseph M. Hellerstein, Ion Stoica, and Raghu Ramakrishnan. Declarative routing: Extensible routing with declarative queries. *SIGCOMM Comput. Commun. Rev.*, 35(4):289–300, August 2005.

[Liu11] Jia Liu. A proof of coincidence of labeled bisimilarity and observational equivalence in applied pi calculus. Technical Report ISCAS-SKLCS-11-05, State Key Laboratory of Computer Science, Institute of Software of Chinese

Academy of Sciences, 2011. Available at http://lcs.ios.ac.cn/~jliu/papers/LiuJia0608.pdf.

[LTV09] Pascal Lafourcade, Vanessa Terrade, and Sylvain Vigier. Comparison of cryptographic verification tools dealing with algebraic properties. In Pierpaolo Degano and Joshua D. Guttman, editors, *Formal Aspects in Security and Trust, 6th International Workshop, FAST 2009, Eindhoven, The Netherlands, November 5-6, 2009, Revised Selected Papers*, volume 5983 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2009.

[Mar94] Stephen Paul Marsh. Formalising trust as a computational concept. Technical report, Ph.D. Thesis, University of Stirling, 1994.

[Maz05] Laurent Mazaré. Satisfiability of dolev-yao constraints. *Electr. Notes Theor. Comput. Sci.*, 125(1):109–124, 2005.

[Mil99] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[MJG+12] Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Rosu. An overview of the MOP runtime verification framework. *STTT*, 14(3):249–289, 2012.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 166–175. ACM, 2001.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[MSS98] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In Aviel D. Rubin, editor, *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29, 1998*. USENIX Association, 1998.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf, 2008.

[NH06a] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theor. Comput. Sci.*, 367(1-2):203–227, 2006.

[NH06b] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theor. Comput. Sci.*, 367(1):203–227, November 2006.

[NJW+13] Samaneh Navabpour, Yogi Joshi, Chun Wah Wallace Wu, Shay Berkovich, Ramy Medhat, Borzoo Bonakdarpour, and Sebastian Fischmeister. Rithm: a tool for enabling time-triggered runtime verification for C programs. In Bertrand Meyer, Luciano Baresi, and Mira Mezini, editors, *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, pages 603–606. ACM, 2013.

[NS94] Moni Naor and Adi Shamir. Visual cryptography. In Santis [San95], pages 1–12.

[OO89] Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 1989.

[OS14] Marek R. Ogiela and Piotr Sulkowski. Improved cryptographic protocol for digital coin exchange. In *Soft Computing and Intelligent Systems (SCIS), 2014 Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium on*, pages 1148–1151, 2014.

[PBP+10] Reema Patel, Bhavesh Borisaniya, Avi Patel, Dhiren R. Patel, Muttukrishnan Rajarajan, and Andrea Zisman. Comparative analysis of formal model checking tools for security protocol verification. In Natarajan Meghanathan, Selma Boumerdassi, Nabendu Chaki, and Dhinaharan Nagamalai, editors, *Recent Trends in Network Security and Applications - Third International Conference, CNSA 2010, Chennai, India, July 23-25, 2010. Proceedings*, volume 89 of *Communications in Computer and Information Science*, pages 152–163. Springer, 2010.

[PH02] Panos Papadimitratos and Zygmunt Haas. Secure Routing for Mobile Ad hoc Networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, 2002.

[PK00] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity - A proposal for terminology. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, volume 2009 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2000.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[PRT04] Elan Pavlov, Jeffrey S. Rosenschein, and Zvi Topol. Supporting privacy in decentralized additive reputation systems. In Christian Damsgaard Jensen, Stefan Poslad, and Theodosis Dimitrakos, editors, *Trust Management, Second International Conference, iTrust 2004, Oxford, UK, March 29 - April 1, 2004, Proceedings*, volume 2995 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2004.

[PS10] Alfredo Pironti and Riccardo Sisto. Provably correct java implementations of spi calculus security protocols specifications. *Computers & Security*, 29(3):302–314, 2010. Special issue on software engineering for secure systems.

[PSD04] Davide Pozza, Riccardo Sisto, and Luca Durante. Spi2java: Automatic cryptographic protocol java code generation from spi calculus. In *18th International Conference on Advanced Information Networking and Applications (AINA 2004), 29-31 March 2004, Fukuoka, Japan*, pages 400–405. IEEE Computer Society, 2004.

[PSW95] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. How to break another provably secure payment system. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 21-25, 1995, Proceeding*, volume 921 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 1995.

[PW91] Birgit Pfitzmann and Michael Waidner. How to break and repair A "provably secure" untraceable payment system. In Joan Feigenbaum,

editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 338–350. Springer, 1991.

[RCR15] Giles Reger, Helena Cuenca Cruz, and David E. Rydeheard. MarQ: Monitoring at runtime with QEA. In *TACAS'15*, pages 596–610, 2015.

[Reg14] Giles Reger. Automata Based Monitoring and Mining of Execution Traces. PhD thesis, University of Manchester, 2014.

[RKZF00] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, 2000.

[RS01] Peter Y. A. Ryan and Steve A. Schneider. Process algebra and non-interference. *Journal of Computer Security*, 9(1/2):75–103, 2001.

[RS11] Mark D. Ryan and Ben Smyth. Applied pi calculus. In *Formal Models and Techniques for Analyzing Security Protocols*, chapter 6. IOS Press, 2011.

[RSG+00] Peter Y. A. Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *The Modelling and Analysis of Security Protocols: The CSP Approach*. Addison-Wesley Professional, 2000.

[RT01] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with finite number of sessions is np-complete. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, page 174. IEEE Computer Society, 2001.

[RZ02] Paul Resnick and Richard Zeckhauser. Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system. In Michael R. Baye (ed.), editor, *The Economics of the Internet and E-commerce (Advances in Applied Microeconomics)*, volume 11, pages 127–157. Emerald Group Publishing Limited, 2002.

[San95] Alfredo De Santis, editor. *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*. Springer, 1995.

[SAR13] Ben Smyth, Myrto Arapinis, and Mark D. Ryan. Translating between equational theories for automated reasoning. *FCS 2013 Workshop on Foundations of Computer Security (Informal Proceedings)*, page 50, 2013.

[SCF+13] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure distributed programming with value-dependent types. *J. Funct. Program.*, 23(4):402–451, 2013.

[Sch97] Berry Schoenmakers. Security aspects of the ecash$^{\text{tm}}$ payment system. In Bart Preneel and Vincent Rijmen, editors, *State of the Art in Applied Cryptography, Course on Computer Security and Industrial Cryptography, Leuven, Belgium, June 3-6, 1997. Revised Lectures*, volume 1528 of *Lecture Notes in Computer Science*, pages 338–352. Springer, 1997.

[SDL+02] Kimaya Sanzgiri, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer. A secure routing protocol for ad hoc networks. In *ICNP*, pages 78–89. IEEE Computer Society, 2002.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SK14] Aye Thandar Swe and Khin Khat Khat Kyaw. Formal analysis of secure e-cash transaction protocol. In *International Conference on Advances in Engineering and Technology, ICAET*, Singapore, 2014.

[SLD+05] Kimaya Sanzgiri, D. LaFlamme, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer. Authenticated routing for ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 23(3):598–610, 2005.

[SMCB12] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In Stephen Chong, editor, *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE, 2012.

[SPP01] Dawn Xiaodong Song, Adrian Perrig, and Doantam Phan. AGVI - automatic generation, verification, and implementation of security protocols. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 241–245. Springer, 2001.

[SRKK10] Ben Smyth, Mark Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In Alessandro Armando and Gavin Lowe, editors, *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security - Joint Workshop, ARSPA-WITS*

2010, Paphos, Cyprus, March 27-28, 2010. Revised Selected Papers, volume 6186 of *Lecture Notes in Computer Science*, pages 146–163. Springer, 2010.

[Ste06] Sandra Steinbrecher. Design options for privacy-respecting reputation systems within centralised internet communities. In Simone Fischer-Hübner, Kai Rannenberg, Louise Yngström, and Stefan Lindskog, editors, *SEC*, volume 201 of *IFIP*, pages 123–134. Springer, 2006.

[Ste08] Sandra Steinbrecher. Enhancing multilateral security in and by reputation systems. In Vashek Matyás, Simone Fischer-Hübner, Daniel Cvrcek, and Petr Svenda, editors, *FIDIS*, volume 298 of *IFIP Advances in Information and Communication Technology*, pages 135–150. Springer, 2008.

[Tur06] Mathieu Turuani. The cl-atse protocol analyser. In Frank Pfenning, editor, *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2006.

[Tur 7] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936-7.

[ZSLL09] Wenchao Zhou, Oleg Sokolsky, Boon Thau Loo, and Insup Lee. *DMaC*: Distributed monitoring and checking. In Saddek Bensalem and Doron Peled, editors, *Runtime Verification, 9th International Workshop, RV 2009, Grenoble, France, June 26-28, 2009. Selected Papers*, volume 5779 of *Lecture Notes in Computer Science*, pages 184–201. Springer, 2009.