On the Verifiability of (Electronic) Exams

Jannik Dreier[2], Rosario Giustolisi[1], Ali Kassem[4], Pascal Lafourcade[3], and Gabriele Lenzini[1]

[1]SnT/Univ. of Luxembourg
[2]Institute of Information Security, ETH Zurich
[3]University d'Auvergne, LIMOS
[4]Université Grenoble Alpes, CNRS, VERIMAG, Grenoble, France

Verimag Research Report n^o TR-2014-2

February 20, 2015

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UJF
Centre Équation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

On the Verifiability of (Electronic) Exams

Jannik Dreier[2], Rosario Giustolisi[1], Ali Kassem[4], Pascal Lafourcade[3], and Gabriele Lenzini[1]

[1]SnT/Univ. of Luxembourg
[2]Institute of Information Security, ETH Zurich
[3]University d'Auvergne, LIMOS
[4]Université Grenoble Alpes, CNRS, VERIMAG, Grenoble, France

February 20, 2015

Abstract

The main concern for institutions that organize exams is to detect when students cheat. Actually more frauds are possible and even authorities can be dishonest. If institutions wish to keep exams a trustworthy business, anyone and not only the authorities should be allowed to look into an exam's records and verify the presence or the absence of frauds. In short, exams should be *verifiable*. However, what verifiability means for exams is unclear and no tool to analyze an exam's verifiability is available. In this paper we address both issues: we formalize several *individual* and *universal verifiability properties* for traditional and electronic exams, so proposing a set of verifiability properties and clarifying their meaning, then we implement our framework in ProVerif, so making it a tool to analyze exam verifiability. We validate our framework by analyzing the verifiability of two existing exam systems – an electronic and a paper-and-pencil system.

Keywords: Exams, Formal Verification, Verifiability, Applied π-Calculus, ProVerif

Reviewers: Pascal Lafourcade, Gabriele Lenzini

How to cite this report:

@techreport {TR-2014-2,
title = {On the Verifiability of (Electronic) Exams},
author = {Jannik Dreier[2], Rosario Giustolisi[1], Ali Kassem[4], Pascal Lafourcade[3], and Gabriele Lenzini[1]

[1]SnT/Univ. of Luxembourg
[2]Institute of Information Security, ETH Zurich
[3]University d'Auvergne, LIMOS
[4]Université Grenoble Alpes, CNRS, VERIMAG, Grenoble, France},
institution = {{Verimag} Research Report},
number = {TR-2014-2},
year = {}
}

Contents

1 Introduction

Not a long time ago, the only way for a student to take an exam was by sitting in a classroom with other students. Today, students can take exams using computers in test centers or even from home and can be graded remotely. This change is possible thanks to computer-aided or computer-based exams, generally called electronic exams (e-exams). E-exams are integrated in Massive Open Online Courses (MOOC), platforms to open a worldwide access to university lectures. E-exams are also trialled at university exams: at the University Joseph Fourier, exams in pharmacy have been organized electronically in 2014 using tablet computers, and all French medicine exams are planned to be managed electronically by 2016 [12].

All such diverse exam and e-exam protocols should provide a comparable guarantee of security and not only against students that cheat, the main concern of exam authorities, but also against other frauds and the frauds perpetrated by the authorities themselves. More or less effective mitigations exist but to really address the matter exams must be *verifiable*. Verifiable exams can be checked for the presence or the absence of irregularities and provide evidence about the fairness and the correctness of their grading procedures. And they should be welcome by authorities since exam verifiability is also about to be transparent about an exam's being compliance with regulations as well as being able to inspire public trust. Specially, some recent scandals [11, 15] show that frauds do not come only from students, but also from exam authorities.

Ensuring verifiability is generally hard and for exams and e-exams one part of the problem lays in the lack of clarity about what verifiability properties they should offer. Another part comes from the absence of a framework to check exams for verifiability. This paper proposes a solution for both.

Contributions. We provide a clear understanding of verifiability for exam protocols and propose a methodology to analyze their verifiability: we define a formal framework where we model traditional paper-and-pencil and electronic exams. We formalize eleven verifiability properties relevant for exams and, for each property, we state the conditions that a sound and complete verifiability test has to satisfy. Following a practice already explored in other domains [2, 3, 6, 18], we classify our verifiability properties into individual and universal, and we formalize them within our framework. Finally, we implement the verifiability tests in the applied π-calculus and we use ProVerif [4] to run an automated analysis. We validate the effectiveness and the flexibility of our framework by modelling and analyzing two different exam protocols: a paper-and-pencil exam currently used by the University of Grenoble, and an internet-based exam protocol called Remark! [14]. We check whether they admit sound and complete verifiabile tests and discuss what exam roles are required to be honest.

Outline. The next section comments the related work. Section 3 provides definitions and models for exam protocols. Section 4 describes and formalizes eleven verifiability properties, and develops a framework of analysis for them. Section 5 validates the framework. Section 6 draws the conclusions and outlines the future work.

2 Related Work

To the best of our knowledge, there is almost no research done on verifiability for exams. A handful number of papers list informally a few security properties for e-exams [5, 13, 19]. Only one offers a formalization [9]. We comment them shortly.

Castella-Roca *et al.* [5] discuss a secure exam management system which is claimed to provide authentication, privacy, correction and receipt fullness, properties that are described informally. Huszti & Pethő [19] refine these notions as security requirements and propose a cryptographic protocol that is claimed to fulfil the requirements, but the claims are only sustained informally. Bella *et al.* [13] comment a list of requirements which are desirable for electronic and traditional exams, and similar to the previous works, they do not formalize the properties they propose. Instead, Dreier *et al.* [9] propose a model for several authentication and secrecy properties in the formal framework of the Applied π-Calculus [1]. No paper outlined above addresses verifiability.

However, verifiability has been studied in other domains than exams, specially in voting and in auctions. In these domains formal models and definitions of security properties exist stably [10, 20, 21]. 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 [3, 18]. 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 [2, 3, 6]. Kremer *et al.* [20] formalize both individual and universal verifiability in the Applied π-Calculus [1]. 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.* [23] use ProVerif to check different verifiability notions that they express as reachability properties, which ProVerif processes natively. In this paper, we also use ProVerif for the analysis, but the model and the definitions here proposed are more general and constrained neither to the Applied π-Calculus nor to ProVerif.

Verifiability for e-auction is studied in Dreier *et al.* [10]. The manner in which they express sound and complete tests for their verifiability properties has been a source of inspiration for what we present here.

Notable notions related to verifiability are *accountability* and *auditability*. Küsters *et al.* [21] study accountability ensuring that, when verifiability fails, one can identify the participant responsible for the failure. They also give symbolic and computational definitions of verifiability, which they recognize as a weaker variant of accountability. However, their framework needs to be instantiated for each application by identifying relevant verifiability goals. Guts *et al.* [16] define auditability as the quality of a protocol that stores sufficient evidence to convince an honest judge that specific properties are satisfied. Auditability revisits the universal verifiability defined in this paper: anyone, even an outsider without knowledge of the protocol execution, can verify the system relying only on the available pieces of evidence.

3 Exam Model

Any exam, paper-and-pencil or electronic, involves at least two roles: the *candidate* and the *exam authority*. The exam authority can have several sub-roles: the *registrar* registers candidates; the *question committee* prepares the questions; the *invigilator* supervises the exam, collects the answers, and dispatches them for marking; the *examiner* corrects the answers and marks them; the *notification committee* delivers the marking.

Exams run generally in phases, commonly four of them: *Registration*, where the exam is set up and candidates enroll; *Examination*, where candidates answer the questions, give them to the authority, and have them accepted officially; *Marking*, where the exam-tests are marked; and *Notification*, where the grades are notified. Usually, each phase ends before the next one begins, an assumption we embrace.

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-tests*) and a set of marks. Three relations link candidates, exam-tests, and marks along the four phases: `Accepted`, `Marked`, and `Assigned`. They are assumed to be recorded during the exam or build from data logs such as registers or repositories.

Definition 1 (Exam) *An exam E is a tuple (I, Q, A, M, α) where I of type $\mathcal{I}$ is a set of candidate identities, Q of type $\mathcal{Q}$ is a set of questions, A of type $\mathcal{A}$ is a set of answers, M of type $\mathcal{M}$ is a set of marks, and α is the set of the following relations:*

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

Definition 1 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) $I_r \subseteq I$ as the set of candidates who registered for the exam (thus, $I \setminus I_r$ are the identities of the unregistered candidates who have taken the exam), and (b) $Q_g \subseteq Q$ as the questions that the question committee has prepared (thus, $Q \setminus Q_g$ are the additional and illegitimate questions that appear in the exam).

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, but it is inappropriate for long open questions. 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.

4 Verifiability Properties

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 $\mathcal{E} \to \texttt{bool}$, 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 2 *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 complementarity 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-test, 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 Table 1 we select six individual (left) and five universal (right) relevant 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 reminder of the section.

Generally speaking, an exam is fully (individual or universal) verifiable when it satisfies all the properties. Of course, on an exam each property can be verified separately to clearly assess its strengths and weaknesses.

Individual Verifiability Properties (I.V.): Here is the candidate that verifies the exam. She knows her identity i, her submitted exam-test 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, that's why it does not appear in the right-hand-side of the equivalent (see Table 1). However, it might be necessary to implement the test depending on the case study.

There is no individual verifiability 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

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

Table 1: Individual and Universal Verifiability

verify, is whether she got the correct questions, and whether she got her test correctly marked. To verify the validity of her question, we propose the property *Question Validity* which ensures that the candidate receives questions 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. To verify that her mark is correct, the candidate can check the property *Marking Correctness* which ensures that the mark received by the candidate is correctly computed on her exam-test. Verifying *Marking Correctness* could e.g. be realized by giving access to the marking algorithm, so the candidate can compute again the mark that corresponds to her exam-test and compare it to the mark she received. As discussed in Section 3, 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 this case, a candidate may wish to verify more properties about her exam test, and precisely that the integrity of the candidate's exam-test is preserved till marking, and that the integrity of the candidate's mark is preserved from delivery till reception. Preserving the integrity of the exam-test and that of the mark is sufficient for the candidate to be convinced that she got the correct mark, provided the examiner follows the marking algorithm correctly.

Each of the remaining four individual properties covers a different step from exam-test submission till mark reception. This allows to identify in which step the error happened in case of failure. The first property, *Exam-Test Integrity*, is to ensure that the candidate's exam-test is accepted and marked as she submitted it without any modification. Running the *Exam-Test Integrity* test after the end of the exam does not invalidate the property since if an exam-test is lost or modified before being marked, it remains modified also after the exam is over. But the event consisting of an exam-test that is first changed before the marking, and then restored correctly after marking, is not captured by *Exam-Test Integrity*. However, such an event can still be detected by verifying *Marking Correctness*. Another property that also concerns the integrity of the exam-test is *Exam-Test Markedness* which ensures that the exam-test submitted by a candidate is marked without modification. Note that if *Exam-Test Integrity* (*i.e.,* the test $\mathtt{ETI}_{\mathtt{IV}}$) succeeds, then *Exam-Test Markedness* (*i.e.,* the test $\mathtt{ETM}_{\mathtt{IV}}$) also succeeds, namely $\mathtt{ETI}_{\mathtt{IV}}(i,q,a,m,p) \Rightarrow \mathtt{ETM}_{\mathtt{IV}}(i,q,a,m,p)$. However, if the test $\mathtt{ETI}_{\mathtt{IV}}$ fails, but the test $\mathtt{ETM}_{\mathtt{IV}}$ succeeds, this would mean that the candidate's exam-test is modified upon acceptance by the authority, but then restored to its correct version before marking. The latter case could be not relevant to the candidate as her exam-test was unmodified when marked; however such an error can be reported to the responsible authority to investigate the problem and see where the error comes from. Moreover, we might have a protocol that does not provide a test for $\mathtt{ETI}_{\mathtt{IV}}$, but a test for $\mathtt{ETM}_{\mathtt{IV}}$, this could depend on the available data at the end of the exam execution. The remaining two properties ensure that the integrity of the mark attributed to a candidate's exam-test by the examiner

is preserved. The property *Mark Integrity* ensures that the mark attributed to a candidate's exam-test is assigned to that candidate by the responsible authority without any modification; and the property *Mark Notification Integrity* ensures that the candidate receives the mark assigned to her by the authority.

Universal Verifiability Properties (U.V.): These 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-test 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 and Mark Notification Integrity are not relevant because the external observer has no knowledge of the questions nor of the markings 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 eligible candidates can submit an exam-test, and this is guaranteed by *Registration*, which ensures that all accepted exam-tests are submitted by registered candidates. An observer may wish to test that all the marks attributed by the examiners to the exam-tests 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. It may be also interested in checking that no exam-test is modified, added, or deleted till the end of the marking phase: this *Exam-Test Integrity*, which ensures that all and only accepted exam-tests are marked without any modification. Another property that could be useful for an observer is *Exam-Test Markedness*. This ensures that all the accepted exam-tests are marked without modification. Thus, if *Exam-Test Integrity* fails but *Exam-Test Markedness* succeeds, then there is at least one extra marked exam-test which is not included in the set of accepted exam-test by the exam authority. Finally, the observer may wish to check that all and only the marks assigned to exam-tests are assigned to the corresponding candidates with no modifications. This is guaranteed by *Mark Integrity*.

5 Validation

We validate our framework and show its flexibility with two different use cases: a paper-and-pencil exam procedure and an internet-based exam protocol. We analyze their verifiability fully. The modeling and the analysis is done in ProVerif. For the full treatment of the case studies, The ProVerif code is available on line[1]. We consider the Dolev-Yao [8] intruder model that is used in ProVerif. Dishonest roles, when needed, are processes controlled by the intruder.

5.1 Use Case # 1: The Grenoble Exam

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:

Registration: All the students of the course are automatically registered as candidates for the exam; they are informed about the exam's date, time and location. EA assigns a fresh pseudonym to each C. The QC, the course's lecturer(s), prepares the questions and hands them to EA.

Examination: After EA authenticates all Cs, 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 and student number in such a way that the corner, when folded, hides them. Each C also writes down visibly their pseudonyms. Then, EA distributes the questions, and the exam begins. At the end, EA collects the exam-tests, checks that all copies have been returned, that all corners are correctly glued, and gives the exam-tests to E.

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

[1] apsia.uni.lu/stast/codes/exams/proverif_ispec15.tar.gz

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

The Alice-Bob notation of Grenoble exam is presented in Figure 1.

Registration
1- $C \rightarrow EA : C$
2- $EA \rightarrow C : C, pseuC$
Examination
3- $QC \rightarrow EA : ques$
4- $EA \rightarrow C : ques$
5- $C \rightarrow EA : hide(C), pseuC, ques, ans, hide(Sig)$
where $hide$ is a function that hide the information and $Sig = sign(C, pseuC, ques, ans)$
Marking
6- $EA \rightarrow E : hide(C), pseuC, ques, ans, hide(Sig)$
7- $E \rightarrow EA : hide(C), pseuC, ques, ans, hide(Sig), mark$
Notification
8- $EA \rightarrow C : C, mark$

Figure 1: Alice-Bob notation of Grenoble exam protocol.

Formal Model

We model the Grenoble protocol in ProVerif. EA, QC, E and the Cs are modeled as communicating processes that exchange messages over public or private channels. They can behave honestly or dishonestly. We detail later when we need private vs. public channels and honest vs. dishonest participants.

Data sets I, Q, A and M are as in Definition 1. Each set is composed by a selection of messages taken from the data generated by the processes, possibly manipulated by the attacker. For example, Q are all the messages that represent a question. Q_g, subset of Q, are all the messages representing a question that are generated by the QC. The exam's relations are also as in Definition 1. `Accepted` contains all the messages $(i, (q, a))$ (*i.e.,* identity and exam-test) that EA has collected. If the EA is honest, it accepts only the exam-tests submitted by registered candidates. `Marked` contains all the messages $(i, (q, a), m)$ (*i.e.,* identity, exam-test, and mark) that the E has generated after having marked the exam-tests. If E is honest, he marks only exam-tests authenticated by EA. `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-test. `Correct` is a deterministic function that outputs a mark for a given exam-test.

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.

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

Table 2: I.V. properties for the Grenoble exam.

Analysis of Individual Verifiability

We model individual verifiability tests as processes in ProVerif, guided by the properties defined in Table 1. 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"* [22], 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 Question Validity.

A complete test receives its input via private channels and by honest participants. The intruder 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 2 reports the result of the analysis. All properties hold (✓) despite the intruder, but often they hold only assuming some roles to be honest : there are attacks otherwise. All properties but Marking Correctness have sound tests (Table 2, 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 intruder 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 2 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`.

Figure 2: Question Validity test for the Grenoble exam.

```
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.
```

In the proof for soundness, we modified the ProVerif code for EA in such way to emit an event `valid` just after the process receives the question from QC and checks its origin-authenticity, 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. All participants are assumed to be honest in this case.

Exam-Test Integrity: The candidate can check the integrity of her exam-test by comparing the the exam-test she submitted to the one marked by the examiner, as she can consult the later after marking. If the exam-test is unmodified after marking, then it is supposed that it is accepted correctly by the exam authority and not modified before marking. The Exam-Test Integrity test is presented in Figure 3. 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 recorded the submitted form without any modification. After that, the test takes the form marked by the examiner, authenticates it and verify that it is equal to the submitted form, 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 and examiner with the verification value ver_t since in practice the candidate takes the forms from them by hand. Also, that in case of completeness the two channels chA and chM, which used to take the forms from the exam authority and examiner respectively, are private channels, while they are public (controlled by the attacker) in case of soundness. ProVerif proves that the Exam-Test 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

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' = ques && ansX' = ans then
event OK
else
event KO.
```

Figure 3: Exam-test Integrity test for Grenoble exam.

Exam-Test Markedness: Similar to Exam-Test Integrity, the candidate can examine her exam-test after marking and check whether it is marked. So, the exam-test markedness test for Grenoble exam is just similar to the exam-test integrity test except that for exam-test markedness the test takes only the marked form. So, the test checks that if the candidate's exact exam-test is marked by a grade. We show by ProVerif that Exam-Test Markedness is complete and sound in case of honest examiner for Grenoble exam.

Marking Correctness: In Grenoble exam when the candidate examine her marked exam-test 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-test 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-test is equal to the received mark. The Marking Correctness test is presented in Figure 4. 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 4: Marking Correctness test for Grenoble exam.

Mark Integrity: As the candidate can examine her exam-test after marking, so she can knows the mark delivered by the examiner on her exam-test. 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 5. 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, =ques, =ans, =hS, 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 5: 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 6. 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. But, 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 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-tests. Thus, he can check whether all accepted exam-tests are submitted by registered candidates; (2) for Exam-Test Markedness, in addition to the accepted exam-tests, he knows the set of marked exam-tests. Then, he can check whether all the accepted exam-tests are marked; (3) for Exam-Test Integrity, he knows the same data as in Exam-Test Markedness. The auditor has to check that all and only the accepted exam-tests are marked; (4) for Marking Correctness, he knows the correction

```
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 6: Mark Notification Integrity test for Grenoble exam.

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

Table 3: U.V. properties for the Grenoble exam.

algorithm and the marked exam-tests together with the delivered marks. The test is to run the correction algorithm again on each exam-test 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 paper.

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 4 depicts the result of the analysis. We must report that in our testing universal verifiability, not for all tests we were able to run 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 all candidates. For these tests we ran ProVerif only for the base case, that where we have only one accepted exam-test 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.

Unfortunately, we were unable to analyze fully automatically some tests in the general case, that is, for any number of participants. This is because ProVerif does not support loops, and we need to iterate over *e.g.,* all candidates. For these tests, we ran ProVerif only for the base case where we have only one accepted exam-test or one assigned mark; then we completed a manual proof that generalizes this result to the general case with an arbitrary number of exam-tests or marks.

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 7.

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

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

holds for any size n of the set `Accepted` and any number m of registered candidates.

Let $\texttt{RV}_\texttt{k}(\cdot)$ denotes the Registration test that can be applied to an exam execution that has k accepted exam-tests, and $\texttt{RV}_\texttt{k}(\cdot) \rightarrow^* OK$ denotes that the test outputs OK (true) after some steps given certain exam execution with k accepted exam-tests.

Let E be an exam execution that has m registered candidates and n accepted exam-tests, and let E_j denotes a version of E where only the j^{th} accepted exam-test 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-test and any number of registered candidates, thus we have for soundness

$$\forall 1 \leq j \leq n : RV_1(E_j) \rightarrow^* OK \Rightarrow i_j \in I_r$$

and for completeness

$$\forall 1 \leq j \leq n : i_j \in I_r \Rightarrow RV_1(E_j) \rightarrow^* OK.$$

The test $RV_n(E)$ checks if each of the accepted exam-tests received on the channels $\texttt{chA1}, \ldots, \texttt{chAn}$ is submitted by one of the registered candidates given on the channels $\texttt{chR1}, \ldots, \texttt{chRn}$, and $\forall 1 \leq j \leq n : RV_1(E_j)$ checks if the j^{th} accepted exam-test received on the channel chAj is submitted by the one of the registered candidates given on the channels $\texttt{chR1}, \ldots, \texttt{chRn}$. Thus, for soundness, we have

$$\begin{array}{c} RV_n(E) \rightarrow^* OK \\ \Downarrow \\ \forall 1 \leq j \leq n : RV_1(E_j) \rightarrow^* OK \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : i_j \in I_r \\ \Downarrow \\ \{i : (i, (q, a)) \in \texttt{Accepted}\} \subseteq I_r \end{array}$$

We can make a similar argument for completeness:

$$\begin{array}{c} \{i : (i, (q, a)) \in \texttt{Accepted}\} \subseteq I_r \\ \Downarrow \\ \forall 1 \leq j \leq n : i_j \in I_r \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : RV_1(E_j) \rightarrow^* OK \\ \Downarrow \\ RV_n(E) \rightarrow^* OK \end{array}$$

Exam-Test Integrity: The Exam-Test Integrity test checks whether all and only the accepted exam-tests are marked. The Exam-Test Integrity test for Grenoble exam is presented in Figure 8. We show, with ProVerif, that Exam-Test Integrity test is complete and sound, with honest exam authority and examiner, for the case of one accepted exam-test and one marked exam-test. Then, we generalize it to the case of any number of accepted exam-tests, more precisely, we show that

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

holds for any size of the sets Accepted and Marked.

Without lost 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-test 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-Test Integrity test that can be applied to an exam execution that has k accepted exam-tests and k marked exam-test, and let $\texttt{ETI}_\texttt{k}(\cdot) \rightarrow^* OK$ denotes that the test outputs OK after some steps for a given exam execution with k accepted and k marked exam-tests. Let E be an exam execution that has n accepted exam-tests and n marked exam-test, and let E_j denotes a version of E where only the j^{th} accepted exam-test (q_j, a_j), which is submitted by the candidate i_j, and j^{th} marked exam-test (q'_j, a'_j), which is related to the candidate i'_j, are counted. Note that, we assume that the exam-tests are marked in the same order as they were accepted.

```
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 7: Universal Registration test for Grenoble exam.

As we show with ProVerif that the test is complete and sound for one exam-test, we have for soundness

$$\forall 1 \leq j \leq n : ETI_1(E_j) \rightarrow^* OK \Rightarrow (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j))$$

and for completeness

$$\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j)) \Rightarrow ETI_1(E_j) \rightarrow^* OK.$$

The $ETI_n(E)$ checks if all the accepted exam-tests received on the channels `chA1`, ..., `chAn` are exactly the marked exam-tests received on channels `chM1`, ..., `chMn`, and $\forall 1 \leq j \leq n : ETI_1(E_j)$ checks if the j^{th} accepted exam-test on the channel `chAj` is exactly the exam-test marked on channel `chMj`. Thus, for soundness, we have

$$\begin{array}{c} ETI_n(E) \rightarrow^* OK \\ \Downarrow \\ \forall 1 \leq j \leq n : ETI_1(E_j) \rightarrow^* OK \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : (i_j, (q_j, a_j)) = (i'_j, (q'_j, a'_j)) \end{array}$$

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

We can make a similar argument for completeness:

$$\texttt{Accepted} = \{(i,(q,a)) : (i,(q,a),m) \in \texttt{Marked}\}$$
$$\Downarrow$$
$$\forall 1 \leq j \leq n : (i_j,(q_j,a_j)) = (i'_j,(q'_j,a'_j))$$
$$\Downarrow$$
$$\forall 1 \leq j \leq n : ETI_1(E_j) \rightarrow^* OK$$
$$\Downarrow$$
$$ETI_n(E) \rightarrow^* OK$$

```
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 8: Universal Exam-Test Integrity test for Grenoble exam.

Exam-Test Markedness: We assume that a third party can takes the marked exam-tests from the examiner, so that he can checks whether all the accepted exam-tests (which he can takes from exam authority) are marked. The Exam-Test Markedness test for Grenoble exam is presented in Figure 9.

We show, with ProVerif, that Exam-Test Markedness test is complete and sound, with honest exam authority and examiner, for the case where we have one accepted exam-test and any number of marked exam-tests. Then, we generalize it to the case of any number of accepted exam-tests, more precisely, we show that

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

holds for any size n of the set `Accepted` and any number m of marked exam-tests.

Let $\texttt{ETM}_\texttt{k}(\cdot)$ denotes the Exam-Test Markedness test that can be applied to an exam execution that has k accepted exam-tests, and $\texttt{ETM}_\texttt{k}(\cdot) \rightarrow^* OK$ denotes that the test outputs OK after some steps given certain exam execution with k accepted exam-tests.

Let E be an exam execution that has n accepted exam-tests and m marked exam-tests, and let E_j denotes a version of E where only the j^{th} accepted exam-test (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-test and any number of marked exam-tests, thus we have for soundness

$$\forall 1 \leq j \leq n : ETM_1(E_j) \rightarrow^* OK \Rightarrow (i_j, (q_j, a_j)) \subseteq \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\}$$

and for completeness

$$\forall 1 \leq j \leq n : (i_j, (q_j, a_j)) \subseteq \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\} \Rightarrow ETM_1(E_j) \rightarrow^* OK.$$

The test $ETM_n(E)$ checks if each of the accepted exam-tests received on the channels chA1, ..., chAn is one of the marked exam-tests received on the channels chM1, ..., chMm, and $\forall 1 \leq j \leq n : ETM_1(E_j)$ checks if the j^{th} accepted exam-test received on the channel chAj is one of the marked exam-tests received on the channels chM1, ..., chMm. Thus, for soundness, we have

$$\begin{array}{c} ETM_n(E) \rightarrow^* OK \\ \Downarrow \\ \forall 1 \leq j \leq n : ETM_1(E_j) \rightarrow^* OK \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : (i_j, (q_j, a_j)) \subseteq \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\} \\ \Downarrow \\ \texttt{Accepted} \subseteq \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\} \end{array}$$

We can make a similar argument for completeness:

$$\begin{array}{c} \texttt{Accepted} \subseteq \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\} \\ \Downarrow \\ \forall 1 \leq j \leq n : (i_j, (q_j, a_j)) \subseteq \{(i, (q, a)) : (i, (q, a), m) \in \texttt{Marked}\} \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : ETM_1(E_j) \rightarrow^* OK \\ \Downarrow \\ ETM_n(E) \rightarrow^* OK \end{array}$$

Marking Correctness: We assume that any one can access the correction algorithm, so a third party can checks whether the delivered marks are computed correctly provided he given an access to the marked exam-tests also. The Marking Correctness test is presented in Figure 10.

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-test, *i.e.,* size of Marked is 1. Then, we generalize it to the case of any number of marked exam-tests, more precisely, we show that

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

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-tests, and $\texttt{MC}_\texttt{k}(\cdot) \rightarrow^* OK$ denotes that the test outputs OK after some steps for a given exam execution with k marked exam-tests. Let E be an exam execution that has n marked exam-tests, and let E_j denotes a version of E where only the j^{th} marked exam-test (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-test, we have for soundness

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

```
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 9: Universal Exam-test Markedness for Grenoble exam.

and for completeness

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

The $MC_n(E)$ checks if all the marked exam-tests received on the channels chM1, ..., chMn are marked correctly, and $\forall 1 \leq j \leq n : MC_1(E_j)$ checks if the j^{th} marked exam-test received on the channel chMj is marked correctly. Thus, we have for soundness,

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

We can make a similar argument for completeness:

```
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 10: Universal Marking Correctness test for Grenoble exam.

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

Mark Integrity: We assume that a third party can takes the list of assigned mark with the corresponding candidates from the exam authority. Thus, simply he can checks 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 11.

We show, with ProVerif, 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

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

holds for any size of the sets `Marked` and `Assigned`.

Without lost 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 outputs 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 $\texttt{MI}_\texttt{k}(\cdot) \rightarrow^* OK$ 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-test, we have for soundness

$$\forall 1 \leq j \leq n : MI_1(E_j) \rightarrow^* OK \Rightarrow (i_j, m_j) = (i'_j, m'_j)$$

and for completeness

$$\forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \Rightarrow MI_1(E_j) \rightarrow^* OK.$$

```
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 11: Universal Mark Integrity test for Grenoble exam.

The $MI_n(E)$ 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 $\forall 1 \leq j \leq n : MI_1(E_j)$ 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 assigned to the correct candidate. Thus, for soundness, we have

$$\begin{gathered} MI_n(E) \rightarrow^* OK \\ \Downarrow \\ \forall 1 \leq j \leq n : MI_1(E_j) \rightarrow^* OK \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \\ \Downarrow \\ \{(i,m) : (i,(q,a),m) \in \texttt{Marked}\} = \texttt{Assigned} \end{gathered}$$

We can make a similar argument for completeness:

$$\begin{gathered} \{(i,m) : (i,(q,a),m) \in \texttt{Marked}\} = \texttt{Assigned} \\ \Downarrow \\ \forall 1 \leq j \leq n : (i_j, m_j) = (i'_j, m'_j) \\ \Downarrow_{(by\ ProVerif)} \\ \forall 1 \leq j \leq n : MI_1(E_j) \rightarrow^* OK \\ \Downarrow \\ MI_n(E) \rightarrow^* OK \end{gathered}$$

Table 4: Universal verifiability properties for the Grenoble exam.

Property	Sound	Complete
Registration	Honest EA	✓
Exam-Test Integrity	Honest EA and E	✓
Exam-Test Markedness	Honest EA and E	✓
Marking Correctness	Honest E	✓
Mark Integrity	Honest EA and E	✓

5.2 Second Use Case: Remark!

The second exam system that we model and analyze is an internet-based cryptographic exam protocol called Remark! [14]. A key feature of the Remark! protocol is the adoption of *exponentiation mixnet* [17] (in short, NET) to generate the pseudonyms for candidates and examiners at registration. The pseudonyms are eventually used as public and verification keys in such a way to allow parties to communicate anonymously. The speciality of the NET is that each server blinds its entries by a common exponent value. Thus, given the entry X, the NET outputs X^r where r is the product of the exponent values of all the servers. The protocol assumes at least one honest server in NET. Another key feature of Remark! is the adoption of an authenticated append-only bulletin board [7] that the protocol employs to publish the pseudonyms, the receipts of test submissions, and the notification of the marks.

Remark! claims to guarantee several authentication, privacy, and verifiability properties. That Remark! ensures the claimed authentication and privacy properties has been proved in [9], but no formal analysis of its verifiability exists. In the context of this paper, such an analysis is a challenging use case. Remark! engages the typical exam roles: the candidate (C), the examiner (E), and the exam authority (EA) (called *manager* in the original paper). We briefly describe the four phases of Remark! below. For the full description we refer to [14]. The corresponding Alice-Bob notation is depicted in Figure 12.

Registration. The list of eligible candidates' and examiners' public keys is sent as a batch to the NET. The NET calculates the pseudonyms (which correspond to public keys that will be used throughout the protocol to allow anonymous encryption and signatures) by raising the initial public keys to a common value $r = \prod_i r_i$ (steps 1 and 4). More specifically, each mix server raises the input message to a secret value r_i, and forwards it to another mix server. At the same time the NET blindly permutates the batch of public keys, which eventually become the pseudonyms for candidates and examiners. Note that all servers at the same stage i of the mix cascade use the same value r_i. Along with the public keys $y' = y^r = (g^x)^r$, the NET publishes a new generator h, which is the output of g raised to the product of each mix server secret value, i.e. $h = g^r$. The NET publishes the pseudonyms and the new generators on the bulletin board (step 2 and 5). Both the candidates and the examiners can identify their own pseudonyms by raising h to their secret key, i.e. $h^x = (g^r)^x$ (steps 3 and 6). Note that two different batches are used for the candidates and examiners, hence they have different random values r_c and r_e.

Examination. The exam authority signs and encrypts the test questions with the candidate's pseudonym (i.e. public key), and publishes them on the bulletin board (step 7). Each candidate submits their answers, which are signed with the candidate's private key (but using h instead of g as the generator to match to his pseudonym), and encrypted with the public key of the exam authority (step 8). The exam authority collects the test answers, checks its signature using the candidate's pseudonym, re-signs them, and publishes their encryption with the corresponding candidate's pseudonym as receipt on the bulletin board (step 9).

Marking. The exam authority then encrypts the signed test answers with an eligible examiner pseudonym (i.e. public key), and publishes each encryption on the bulletin board (step 10). The corresponding examiner grades the test answers, and signs them with his private key (again using h instead of g as the generator

Assumption: The protocol assumes a list of eligible examiners and their public keys PK_E, and a list of eligible candidates and their public keys PK_C.

Examiner Registration
1- NET calculates $\overline{r}_e = \prod_{i=1}^{k} r_{e_i}$, $\overline{PK}_E = PK_E^{\overline{r}_e}$ and $h_e = g^{\overline{r}_e}$
2- NET publishes $sign((\overline{PK}_E, h_e), SK_{NET})$
3- E checks if $\overline{PK}_E = h_e^{SK_E}$

Candidate Registration
4- NET calculates $\overline{r}_c = \prod_{i=1}^{k} r_{c_i}$, $\overline{PK}_C = PK_C^{\overline{r}_c}$ and $h_c = g^{\overline{r}_c}$
5- NET publishes $sign((\overline{PK}_C, h_c), SK_{NET})$
6- C checks if $\overline{PK}_C = h_c^{SK_C}$

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

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

Notification
12- $EA \rightarrow C : \{M_a, sign(M_a)_{\overline{SK_E, h_e}}\}_{\overline{PK}_C}$
13- $NET \rightarrow EA : \{\overline{r}_c, sign(\overline{r}_c)_{SK_N}\}_{PK_{EA}}$

Figure 12: The Remark! e-exam protocol

to match to his pseudonym). The examiner then encrypts them with the exam authority public key, and submits their marks to the exam authority (step 11).

Notification. When the exam authority receives all the candidate evaluations, it publishes the signed marks encrypted with the corresponding candidate's pseudonym (step 12). Then, the exam authority asks the NET to de-anonymize the candidate's pseudonyms by revealing their secret exponents (step 13). In so doing, the candidate anonymity is revoked, and the mark can finally be registered. Note that the examiners secret exponent is not revealed to ensure their anonymity even after the exam has ended.

Formal Model

We model Remark! in ProVerif. The roles C, E, EA and NET are modelled as communicating processes, while the BB is modelled as a public channel. The equational theory that models the capabilities of the pseudonym, in addition to the classical cryptographic primitives, is reported in Table 5. Note that the encryption with pseudonyms preserves the probabilistic feature of ElGamal encryption.

Data sets I, Q, A and M are as in Definition 1. We populate the set I with the C's pseudonyms rather than their identities. This replacement is sound because C is uniquely identified by her key, and the equational theory preserves the bijective mapping between keys and pseudonyms. The sets Q, A, and M are the messages that correspond to the questions, the answers and the marks generated by the protocol's parties.

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 publish on BB at Examination.

The tuples $(i, (q, a), m)$ and (i, m) of `Marked` and `Assigned` respectively consist of the posts that EA publish on BB at Marking. Precisely, the tuple $(i, (q, a), m)$ is built from the marked exam-test signed by E, while the tuple (i, m) is built from the encryption of the marked exam-test 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-test.

Table 5: Equational theory to model Remark!

$$\begin{aligned}
\texttt{checkpseudo}(\texttt{pseudo_pub}(\texttt{pk}(k), rce), \\
\texttt{pseudo_priv}(k, \texttt{exp}(rce))) &= \texttt{true} \\
\texttt{decrypt}(\texttt{encrypt}(m, \texttt{pk}(k), r), k) &= m \\
\texttt{decrypt}(\texttt{encrypt}(m, \texttt{pseudo_pub}(\texttt{pk}(k), rce), r), \\
\texttt{pseudo_priv}(k, \texttt{exp}(rce))) &= m \\
\texttt{getmess}(\texttt{sign}(m, k)) &= m \\
\texttt{checksign}(\texttt{sign}(m, k), \texttt{pk}(k)) &= m \\
\texttt{checksign}(\texttt{sign}(m, \texttt{pseudo_priv}(k, \texttt{exp}(rce))), \\
\texttt{pseudo_pub}(pk(k), rce)) &= m
\end{aligned}$$

Finally, `Correct` is the algorithm used to mark the exam-tests, which is modelled as 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! originally mandates only two individual verifiability properties: Mark Notification Integrity and a weaker version of Exam-Test Integrity. However, we checked which assumptions Remark! needs in order to ensure all our properties. For each property, we describe how to build the corresponding test.

Question Validity. The test (Figure 13) 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 an the event `generated`. ProVerif shows that Remark! is question validity verifiable.

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

Figure 13: The Question Validity test for Remark!.

Exam-Test Integrity. We note that Remark! is Exam-Test Integrity testable. Given the exam-test submitted by the candidate (`ques`,`ans`), the corresponding receipt (`eca'`), and the notification (`ema'`)published on the bulletin board by the exam authority, the test (Figure 14) checks if they (*i.e.,* submission, receipt, and notification) contain the same question, answer, 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.

Note that a dishonest exam authority can publish two different receipts for the same exam-test on the bulletin board. However, since the bulletin board is append-only, the candidate notices if the exam authority appends two different receipts for her submission because only the candidate knows the private key.

ProVerif shows that the test for Exam-Test Integrity is sound and complete.

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

let (ca:bitstring, sca':bitstring)=decrypt(eca', priv_C) in
let (ques':bitstring, ans':bitstring, pseudo_C':pkey)=
    checksign(sca', pkA) in
let (((ques'':bitstring, ans'':bitstring, pseudo_C'':pkey),
      sca1:bitstring, mark:bitstring), sma:bitstring)=
    decrypt(ema', priv_C) in
let ((ques''':bitstring, ans''':bitstring, pseudo_C''':pkey),
     sca1':bitstring, mark':bitstring)=
    checksign(sma, pseudo_E) in
if  ques'=ques && ans'=ans && pseudo_C'=pseudo_C &&
    ques''=ques && ans''=ans && pseudo_C''=pseudo_C &&
    (ques', ans', pseudo_C')=checksign(sca1,pkA) &&
    ques'''=ques && ans'''=ans && pseudo_C'''=pseudo_C &&
    sca1'=sca1 then event OK
else event KO.
```

Figure 14: The Exam-test Integrity test for Remark!.

Exam-Test Markedness. We note that if the test for Exam-Test Integrity returns `OK` then also the test for Exam-Test Markedness returns `OK`. In this case the test input does not include the receipt, and the test checks if submission and notification contain the same question, answer, and pseudonym.

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 (both soundness and completeness). Given the exam-test submitted by the candidate and the table of the evaluations, the test (Figure 15) checks if the mark reported on table that corresponds to the exam-test 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':bitstring) in
if mark'=mark then event OK
else event KO.
```

Figure 15: The Marking Correctnessfor Remark!.

Mark Integrity. Similarly to Exam-Test Integrity, we can show that Remark! is mark integrity verifiable. Given the exam-test submitted by the candidate (`ques`,`ans`) and the corresponding notification (`ema'`)published by the exam authority, the test (Figure 16) checks if they contain the same question, answer, and pseudonym and that the examiner's signature on the mark is correct.

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

Table 6: I.V. properties for Remark!

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 include the signature of the examiner.

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

let (((ques':bitstring, ans':bitstring, pseudo_C':pkey),
      sca':bitstring, mark:bitstring), sma:bitstring)=
    decrypt(ema', priv_C) in
let ((ques'': bitstring, ans'': bitstring, pseudo_C'': pkey),
     sca'': bitstring, mark': bitstring)=
    checksign(sma, pseudo_E) in
if ques'=ques && ans'=ans && pseudo_C'=pseudo_C &&
   ques''=ques && ans''=ans && pseudo_C''=pseudo_C && mark=mark' &&
   (ques', ans', pseudo_C')=checksign(sca',pkA) then event OK
else event KO.
```

Figure 16: The Mark Integrity test for Remark!.

Mark Notification Integrity. It is easy to build a test for Mark Notification Integrity. Given the mark notified to the candidate and the corresponding notification (`ema'`) published by the exam authority, the test (Figure 17) 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-Test Integrity is sound and complete.

```
let testMNI (pkA, priv_ch, bba)=
in(priv_ch, (priv_C:skey, mark:bitstring, pseudo_C:pkey));
in(bba, ema':bitstring);
let (((ques:bitstring, ans:bitstring, pseudo_C':pkey),
      sca:bitstring, mark':bitstring), sma:bitstring)=
    decrypt(ema, priv_C) in
if (ques,ans,pseudo_C')=checksign(sca, pkA)
&& pseudo_C'=pseudo_C && mark'=mark then event OK
else event KO.
```

Figure 17: The Mark Notification Integrity test for Remark!.

Table 6 resumes the ProVerif results of our analysis.

Analysis of Universal verifiability

We verify most 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 needed. However, an auditor can access some data posted by EA after the exam concludes. Precisely, we assume 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-Test Integrity, the EA reveals the marked exam-test 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-Test Markedness, the auditor accesses the same data outlined above for Exam-Test Integrity. However, since Remark! is exam-test integrity universally verifiable, it is easy to show that the protocol is exam-test markedness universally verifiable, too; (4) for Marking Correctness, the EA reveals the marked exam-test, the random values used to encrypt the marked exam-test, and a table that maps a mark to each answer, after the exam concludes. According our test codes outlined in Figures 18-19, 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 5.1 for the case with more candidates. Table 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 18, 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-Test Integrity We prove that Remark! is exam-test 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 19 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-tests.

Exam-Test Markedness Since Remark! is exam-test integrity universally verifiable, it is easy to show that it is also exam-test 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 20, 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-Test Integrity, we assume that one examiner marks all the exam-tests for simplicity.

Mark Integrity Regarding this property, we note that is the input needed for this test (Figure 21) 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-Test Integrity	✓	✓(all)
Exam-Test Markedness	✓	✓(all)
Marking Correctness	✓ (EA)	✓(all)
Mark Integrity	✓	✓(all)

Table 7: U.V. properties for Remark!

6 Conclusion

This paper studies verifiability for exam protocols, a security feature that has been studied for voting and auctions but not for exams. In this domain, verifiability properties revealed to be peculiar and required novel definitions. We defined several properties which we organized in individual and universal verifiability properties; moreover, we developed a formal framework to analyze them. As far as we know, we are the first to have developed a framework for verifiability of exam protocols.

Our properties and our methodology work for both to electronic (*e.g.,* cryptographic) exam protocols and to traditional (*e.g.,* paper-and-pencil) exams. Thus, in addition to cryptographic protocols, our methodology is applicable to systems that handle physical security measures, such as a face-to-face authentication. In ProVerif, where we implement our framework, such communications are tricky and we developed ad-hoc equational theories to capture their properties.

We have validated our framework by analyzing the verifiability of two existing exams, one paper-and-pencil-based and the other Internet-based. We run most of the tests automatically; but where ProVerif could not handle the verification of the general case for some universal verifiability properties, we proved manually the tests by induction.

The paper-and-pencil exam has been proved to satisfy 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.

The result of the analysis of the Internet-based protocol are somehow complementary. All properties but three are sound without assuming that the exam's roles are honest. But *Marking Correctness*, which worked without assumption in the paper-and-pencil case, 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.

Since the interest in exam protocols is growing, as future work we plan to analyze more protocols and corroborate more extensively the validity of our framework. Another future work regards the use of tools. Since ProVerif's equational theories, used to model cryptographic primitives, introduce an abstraction which may bring the analyst to miss attacks, we intend to trail CryptoVerif [**?**] and achieve stronger proofs. In a framework extended with the new tool, we plan to show how to set up an analysis based on the computational model.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL'01*, pages 104–115, New York, 2001. ACM. 2

```
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 18: The universal Registration test for Remark!.

```
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 19: The universal Exam-Test Integrity test for Remark!.

```
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 20: The universal Marking Correctness test for Remark!.

```
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 21: The universal Mark Integrity test for Remark!.

[2] J. Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, December 1996. 1, 2

[3] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC '94*, pages 544–553, New York, NY, USA, 1994. ACM. 1, 2

[4] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW*, pages 82–96, Cape Breton, Canada, June 2001. IEEE Computer Society. 1

[5] J. Castellà-Roca, J. Herrera-Joancomartí, and A. Dorca-Josa. A Secure E-Exam Management System. In *ARES*, pages 864–871. IEEE Computer Society, 2006. 2

[6] J. Cohen and M. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *FOCS'85*, pages 372–382, Portland, Oregon, USA, October 1985. IEEE Computer Society. 1, 2

[7] C. Culnane and S. Schneider. A peered bulletin board for robust use in verifiable voting systems. In *CSF*, 2014. 5.2

[8] D. Dolev and A. C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983. 5

[9] J. Dreier, R. Giustolisi, A. Kassem, P. Lafourcade, G. Lenzini, and P. Y. A. Ryan. Formal analysis of electronic exams. In *SECRYPT'14*. SciTePress, 2014. 2, 5.2

[10] J. Dreier, J. H., and P. Lafourcade. Defining verifiability in e-auction protocols. In *ASIACCS'13*, pages 547–552. ACM, 2013. 2

[11] E. Flock. Aps (atlanta public schools) embroiled in cheating scandal. Retrieved from http://www.washingtonpost.com/blogs/blogpost/post/aps-atlanta-public-schools-embroiled-in-cheating-scandal/2011/07/11/gIQAJl9m8H_blog.html, November 2011. 1

[12] U. J. Fourier. Premier examen sur tablettes numériques. Press release, Februrary 2014. Available at http://www.ujf-grenoble.fr/universite/medias-et-communication/actualites/premier-examen-sur-tablettes-numeriques-1521820.htm. 1

[13] R. Giustolisi, G. Lenzini, and G. Bella. What security for electronic exams? In *CRiSIS'13*, pages 1–5. IEEE, 2013. 2

[14] R. Giustolisi, G. Lenzini, and P. Ryan. Remark!: A secure protocol for remote exams. In *Security Protocols XXII*, LNCS. Springer, 2014. to appear. Preprint http://orbilu.uni.lu/handle/10993/16527. 1, 5.2

[15] F. Guénard. *La Fabrique des Tricheurs: La fraude aux examens expliquée au ministre, aux parents et aux professeurs*. Jean-Claude Gawsewitch, 2012. 1

[16] N. Guts, C. Fournet, and F. Zappa Nardelli. Reliable evidence: Auditability by typing. In *ESORICS'09*, volume 5789 of *LNCS*, pages 168–183. Springer, 2009. 2

[17] R. Haenni and O. Spycher. Secure internet voting on limited devices with anonymized dsa public keys. In *EVT/WOTE'11*. USENIX, 2011. 5.2

[18] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT'00*, volume 1807 of *LNCS*, pages 539–556. Springer, 2000. 1, 2

[19] A. Huszti and A. Pethõ. A secure electronic exam system. *Publicationes Mathematicae Debrecen*, 77:299–312, 2010. 2

[20] S. Kremer, M. Ryan, and B. Smyth. Election verifiability in electronic voting protocols. In *ESORICS'10*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010. 2

[21] R. Küsters, T. Truderung, and A. Vogt. Accountability: definition and relationship to verifiability. In *CCS'10*, pages 526–535. ACM, 2010. 2

[22] P. Y. A. Ryan, S. A. Schneider, M. Goldsmith, G. Lowe, and A. W. Roscoe:. *The Modelling and Analysis of Security Protocols: The CSP Approach*. Addison-Wesley Professional, first edition, 2000. 5.1

[23] B. Smyth, M. Ryan, S. Kremer, and K. Mounira. Towards automatic analysis of election verifiability properties. In *ARSPA-WITS'10*, volume 6186 of *LNCS*, pages 146–163. Springer, 2010. 2