Privacy-Preserving Verifiability: A Case for an Electronic Exam Protocol

Rosario Giustolisi[1], Vincenzo Iovino[2] and Gabriele Lenzini[2]
[1] *IT University of Copenhagen*
[2] *University of Luxembourg - Interdisciplinary Centre for Security, Reliability and Trust (SnT)*
rosgr@itu.dk, {vincenzo.iovino,gabriele.lenzini}@uni.lu

Keywords:
Privacy-Preserving Verifiability, Electronic Exam Protocols, Analysis of Security and Privacy

Abstract:
We introduce the notion of *privacy-preserving verifiability* for security protocols. It holds when a protocol admits a verifiability test that does not reveal to the verifier about the protocol's execution more than it needs to know to run the test. Our definition of privacy-preserving verifiability is general and applies to cryptographic protocols as well as human security protocols; however, in this paper we exemplify it in the domain of e-exam systems. We prove that the notion is meaningful: we study an existing exam protocol that is verifiable but lacking verifiability tests which are privacy-preserving. We prove that the notion is applicable: we extend the same protocol using functional encryption so that it admits a verifiability test that preserves privacy according to our definition. We further verify in ProVerif that the verifiability holds despite malicious parties and that the new protocol maintains all the security properties that the original protocol enjoyed, so proving that our privacy-preserving verifiability can be achieved from existing security.

1 Introduction

'Being verifiable' is an appreciated quality for a security protocol. It says that, during its execution, the protocol safely stores pieces of information that can be used later as evidence to determine that certain properties hold on the protocol's run. A voting system that offers its voters to "verify that their votes have been cast as intended", for instance, generates data that allow voters to test that their votes have been registered and that it contains, with no mistake, the expression of their vote (Adida and Neff, 2006).

This paper is about verifiability, but it also discusses a new requirement for it. The requirement states that a curious verifier, in addition to the verifiability test's result, should learn no more about the system's execution than the pieces of information that he needs to know run the test.

The need-to-know principle is not new (e.g., see (Department of Defence, 1987)). Not new is also the particular interpretation of the principle that link to techniques like zero knowledge proofs (De Santis et al., 1988) where a verifier is convinced of a given statement's truth without learning no more than that. However, in this paper we originally present and discuss the need-to-know principle as a requirement for verifiability tests.

Our notion of *privacy-preserving verifiability* test needs to be formally defined, and we do so in § 3. We prove that our notion of privacy-preserving verifiability is meaningful in § 4. Here, we refer to the particular domain of *electronic exams*, where verifiability and privacy are sensitive properties (the first should work despite untrusted authorities, the second is not everlasting but conditional). We show for a particular protocol, called Remark! (Giustolisi et al., 2014), that we know well and which we have proved verifiable in (Dreier et al., 2014) that several of its current universal verifiability are not privacy-preserving according to our definition.

In § 5, we give evidence that achieving privacy-preserving verifiability is possible. In particular, again taking Remark! as an example, we demonstrate that by modifying the protocol's design it becomes possible to have for one of the proto-

col's verifiability property a verifiability test that is now privacy-preserving. We use functional encryption (Boneh et al., 2011) for this purpose but other viable solutions may be possible. Possible is also to extend privacy-preserving verifiability to the protocol's other verifiability properties.

In § 6, we verify with Proverif (Blanchet, 2014), a security protocol verifier, that the new design does not compromise any of the protocol's enjoyed security properties including all its other verifiability properties. This analysis is a significant step since we prove that privacy-preserving verifiability for a specific verifiability test can be achieved while preserving all the existing security properties of the protocol.

Before moving to the technical part of the paper, it is important to anticipate that private-preserving verifiability can be potentially achieved in different ways, using one of the techniques that computer security research offers today for minimizing the disclosure of information. Non-interactive Zero-knowledge proofs (De Santis et al., 1988) and functional encryption schemes (Boneh et al., 2011) are two techniques that stand out for our purposes. As anticipated above, our privacy-preserving verifiable exam protocol makes use of the second technique (recalled in § 5.1). We do not have a proof here, but we argue in § 7 that only functional encryption has features that make it suitable to our purpose.

2 Related Work

Privacy and verifiability, apparently two contrasting properties, have been discussed in the literature mainly to show that they can be both satisfied in the same system.

Most of the work in this sense is about voting systems, and the literature is vast in this domain. To make our point, we comment the work of Cuvelier *et al.* (Cuvelier et al., 2013). The authors propose a new primitive, called *Commitment Consistent Encryption*, with which they redesign a voting scheme for an election that is verifiable while ensuring everlasting ballot privacy. The problem they solve for voting is clearly related to that we raise here: avoid that from a public audit trail a verifier could learn more than it is allowed to learn. To a certain extent, Cuvelier *et al.* propose a cryptographic instance of the problem that we formalize in this paper. However our notion of privacy-preserving verifiability is more general and not necessarily cryptographic: it applies to cryptographic protocols as well as to non-crypto protocols. It is, in other words, a functional requirement for the verifiability tests that a system offers to auditors. In this sense, our work relates more to the plentiful research that discusses requirements and definitions of privacy. Again, the literature on this subject is huge, but the work that most closely represents the notion of privacy-preservation that we advance here is that of Mödersheim *et al.* (Mödersheim et al., 2013). The work presents and discusses $\alpha-\beta$ privacy where α is what an adversary knows, and β are the cryptographic messages that the adversary sees. Then $\alpha-\beta$ privacy means that the intruder can derive from β only what he can derive from α already. We have not tried it, but we think that our notion of privacy-preservation can be formulated in term of $\alpha-\beta$ privacy, which would be definitely an interesting task since Mödersheim *et al.* proves that their definition subsumes static equivalence of frames and is in some part decidable with existing formal method tools.

Verifiability is a property that has been studied prevalently, but not exclusively, in secure voting. Different models and requirements have been proposed (Küsters et al., 2010; Kremer et al., 2010). In voting, *individual verifiability* signifies that votes have been "cast as intended", "recorded as cast", and "counted as recorded" (Benaloh and Tuinstra, 1994; Hirt and Sako, 2000), while *universal verifiability* signifies that auditors can verify the correctness of the tally using only public information (Cohen and Fischer, 1985; Benaloh and Tuinstra, 1994; Benaloh, 1996). Kremer *et al.* (Kremer et al., 2010) formalised both individual and universal verifiability in the applied pi-calculus, and Smyth *et al.* (Smyth et al., 2010) used ProVerif to check verifiability in three voting protocols.

We base our definition of verifiability on a formalization proposed in a previous work by some of the authors of this paper (Dreier et al., 2015) This choice is not a coincidence. That work is about verifiable e-exams, the same domain that we take here as a use case to demonstrate that our notion of privacy-preserving verifiability is meaningful and effective. Besides, that work proves that Remark! (Giustolisi et al., 2014), an exam protocol rich in security properties, is verifiable. Such a proof is fundamental for this paper since we need to consider a protocol that is verifiable, although we need it be not privacy-preserving. Thus, as use case, we also chose Remark!.

Although contextualized in reference to exams, our research is not bound to work in that domain only. The notion of privacy-preserving verifiability is abstract, and the solution that we propose to ensure privacy-preservation is demonstrated for a universal verifiability test of a common integrity property; it seems plausible to apply our results in other domains, like voting or auction. Proving this claim is future work.

3 Concepts, Models & Definitions

We contextualize our study in the specific domain of exams. Here, we recall what an exam is and discuss an abstract model for protocols in this family that will be used to define the two key concepts of this paper: *verifiability test*, and *privacy-preserving verifiability test*.

An Exam's Anatomy In real exams,there are usually involved several roles. For the goal of this work, it is sufficient to consider only three fundamental roles: the *candidate*, the *exam authority*, who helps in the organization/execution of the exam, and the *examiner*, who grades the exam tests.

An exam's workflow is rigidly structured and organized in at least four distinct sequential phases. At *preparation*, an exam's instance is created, the questions are selected, and the candidates enroll; At *examination*, the candidates sit and take the exam i.e., they are identified and checked for being eligible, receive the test sheet(s), answer the questions, and quit or submit their answers for marking; At *marking*, the exam-tests are assessed and marked; At *notification*, grades are notified and registered.

An Exam's Abstract Model An exam and its execution can be modelled concisely using sets. The definitions below, slightly simplified, comes from (Dreier et al., 2015)

Definition 1. *An* exam *is a tuple* (ID, Q, A, M) *of finite sets: ID are the candidates; Q are all the possible questions; A the possible answers; M all the possible marks.*

Definition 2. *Given an exam* (ID, Q, A, M)*, an exam's* execution*,* $\mathcal{E}$*, is a tuple of two sets*

- $ID' \subseteq ID$ *are the registered candidates;*
- $Q' \subseteq Q$ *are the official questions of the exam;*

and of four relations over the exam

- $\texttt{AssignedQuestions} \subseteq (ID \times Q)$*, links a question to a candidate, supposedly the question the candidate receives by the authority;*
- $\texttt{SubmittedTests} \subseteq (ID \times (Q \times A))$*, links an answered test to a candidate, supposedly the test submitted for marking by the candidate;*
- $\texttt{MarkedTests} \subseteq (ID \times (Q \times A) \times M)$ *links a mark to a exam-tests of a candidate, supposedly the grade given to the test;*
- $\texttt{NotifiedMarks} \subseteq (ID \times M)$ *links a mark to a candidate, supposedly the mark notified to the candidate;*

These four relations model the state of the exam's execution at the end of each of the four exam phases, like if they were execution logs. There are no requirements between the relations, and this is because the model should also express executions which are erroneous or corrupted. It may happen that a test assigned to no one is submitted fraudulently, or that a test assigned to a candidate is duplicated, answered in two possible ways, and both copies are submitted (i.e., creation/duplication of a test). It may happen as well as that a submitted test is never marked because lost (i.e., loss of a legitimately submitted and answered test).

Authentication and integrity properties are expressed as predicates over $\mathcal{E}$. Let c, q, a and m range over ID, Q, A and M respectively. An example of property is *Marking Integrity*, which states that no mark has changed from marking to notification, and it is written as follows:

$$\texttt{NotifiedMarks} \subseteq \{(c, m) : (c, (q, a), m) \in \texttt{MarkedTests}\}$$

Another property, *Testing Integrity* expresses that all the exams that have been marked are submitted for marking. It is as follows:

$$\{(q, a) : (c, (q, a), m) \in \texttt{MarkedTests})\} \subseteq \{(q, a) : (c, (q, a)) \in \texttt{SubmittedTests}\}$$

Properties can be easily combined. For example, *Marking and Test Integrity* is obtained by conjunction from the previous properties:

$$\texttt{NotifiedMarks} \subseteq \{(c, m) : (c, (q, a), m) \in \texttt{MarkedTests} \text{ and } (c, (q, a)) \in \texttt{SubmittedTests}\} \quad (1)$$

Definition 2 serves well also to express properties of verifiability of a security property p. An exam protocol is p-verifiable if it has a *test* for p, that is, an algorithm $\texttt{test}_p$ that given any exam's

execution $\mathcal{E}$ and some knowledge K (e.g., decryption keys) returns true if and only if p holds on $\mathcal{E}$. Formally, p-verifiability can be expressed as

$$\forall \mathcal{E},\ \mathtt{test}_p(\mathcal{E}, K) \text{ iff } \mathcal{E} \models p \tag{2}$$

where $\mathcal{E} \models p$ means that p holds on $\mathcal{E}$.

A Refined Exam's Abstract Model Dreier *et al.*'s model partially serves the goal of this paper but is too abstract to capture non-trivial notions of privacy like the one we intend to advance (see Definition 4). We need a formalism where to express logs that can be encrypted, tests which run over them, and knowledge about the exam in clear which is gained from running the test. It is in the interplay between these elements that our notion of privacy-preserving verifiability emerges.

Thus, besides the *abstract model* $\mathcal{E}$, which represents an exam logs always in clear text, we consider also a *concrete model*, $\{\mathcal{E}\}$ that represents the actual and possibly encrypted logs of an exam's execution. A verifiability test is executed on the concrete model $\{\mathcal{E}\}$.

Definition 3. *Let $\mathcal{E}$ and $\{\mathcal{E}\}$ be an abstract and a concrete exam's execution respectively, and let p be a security property over $\mathcal{E}$. Then, p is verifiable if there exists a test for p, $\mathtt{test}_p$, that given some knowledge K satisfies the following condition:*

$$\forall \mathcal{E},\ \mathtt{test}_p(\{\mathcal{E}\}), K) \textit{ iff } \mathcal{E} \models p \tag{3}$$

Any proof aiming at convincing that p is verifiable needs to clarify how the test that operates on $\{\mathcal{E}\}$ is sound and complete with respect to the validity of p on $\mathcal{E}$.

The notion of privacy-preserving verifiability is defined in respect to what a verifier, say v, can learn about $\mathcal{E}$ by running $\mathtt{test}_p(\{\mathcal{E}\}, K)$. We are, however, not interested to capture what v can learn from the outputs of the test. Potentially, a lot of information can be deduced by inference attacks, see (Naveed et al., 2015), but the leaks due to such attacks are outside the goal of this paper. Rather, we intend to capture what v can learn about $\mathcal{E}$ from the information in $\{\mathcal{E}\}$ he can access while running $\mathtt{test}_p(\{\mathcal{E}\})$.

Let $[\{\mathcal{E}\}]_p$ be the portion of $\{\mathcal{E}\}$ to which a verifier needs to access to run $\mathtt{test}_p(\{\mathcal{E}\})$. Since $[\{\mathcal{E}\}]_p$ can be encrypted, v does not necessarily learn about the exam's execution $\mathcal{E}$. It depends on what v can do from knowing $[\{\mathcal{E}\}]_p$ and K. If we assume v be a passive but curious adversary, and the learning process working at the symbolic level of messages (i.e., not of bits), the deduction relation $\vdash$ over messages defines the ability to learn. It is the minimal relation that satisfies the following equations:

$$\begin{array}{rcl} m_k, k^{-1} & \vdash & m \\ (m, m') & \vdash & m \text{ and } m' \end{array}$$

In the relation above, we abstract from symmetric/asymmetric encryption: k^{-1} is the key to decrypt, whatever the paradigm of reference.

Definition 4. *Let $\mathcal{E}$ be any execution of an exam protocol and $\mathtt{test}_p(\cdot)$ a test for verifiability for a property p. The test is* privacy-preserving *if by running it v does not learn more about the exam execution than he knows from the information he is given to run the test. Formally:*

$$\{m \ :\ ([\{\mathcal{E}\}]_p \cup K) \vdash m\} \cap \mathcal{E} = ([\{\mathcal{E}\}]_p \cup K) \cap \mathcal{E}$$

Definition 4 says that v should learn about $\mathcal{E}$ no more than what he knows already from the portion of the exam's log he needs to access to run the test and from the additional knowledge he needs to know to run the test.

In Definition 4 we have slightly overloaded the operator $\cap$. Since $\mathcal{E}$ is a tuple of sets it must be considered applied over each element of the tuple that is: $(A, B) \cap C = C \cap (A, B) = (A \cap C, B \cap C)$. Similarly the interpretation of $\subseteq$ when applied to tuple of sets, must be extended to tuple, that is: $(A, B) \subseteq (C, D)$ iff $A \subseteq C$ and $B \subseteq D$.

Definition 5 (Privacy-Preserving Verifiability)**.** *A protocol is* privacy-preserving verifiable *with respect to a property p, if p is verifiable and admits a verifiability test $\mathtt{test}_p$ that is privacy-preserving.*

4 Meaningfulness

Claim 1. *Our notion of privacy-preserving verifiability is meaningful: it can be used to distinguish verifiable protocols that are privacy-preserving from those which are not.*

To prove the claim, we refer to an e-exam protocol whose verifiability (with respect to several properties) has been established formally. The protocol is called Remark!. The protocol is structured in the four typical phases of an exam. Its message flow is recalled in Figure 1. In the chart C is a candidate and E is an examiner. Only two roles are presented for conciseness, but the protocol is supposed to work for all candidates and all examiners and it is executed by them all. An exam authority, A, helps the process.

Description The *registration* is the phase where pseudonyms are generated to ensure a double blind anonymity in testing and marking. Here, *n exponential mixnets* generate the pseudonyms Pk_{C_i} for the registered candidate C. The candidate already possesses a pair of public/private keys, $\langle \mathsf{Pk}_C, \mathsf{Sk}_C \rangle$ and uses the private key to recognize its designated pseudonym among the pseudonyms generated for all the candidates and posted on a public bulletin board ($\mathcal{BB}$, in Figure). Similarly, the mixnets generate pseudonyms for the examiners. A zero-knowledge proof of the generation is present on the bulletin board as proof of correctness of the generation process.

At *testing*, the selected questions are signed by the authority and encrypted with the pseudonym of the candidates. The questions are posted on the bulletin board from where the candidates retrieve them. Then they answer the questions. In Figure, *ques* is the question and *ans* is an answer. The candidate C prepares a tuple $\langle ques, ans, \mathsf{Pk}_C \rangle$, which he signs with its private key Sk_C. The tuple is then encrypts with the public key of the authority and send to it. The authority decrypts and re-encrypts the message with the candidate pseudonym and publishes the message on the bulletin board.

At *marking* the authority dispatches the tests to the examiners. In Figure, A signs the answered test and uses the pseudonyms of the examiner, i.e., Pk_E, as encryption keys. He re-encrypts with it the answered test and publishes it on the bulletin board. From there, similarly to what the candidates did with the questions, E retrieves the test, marks them (i.e., assigns a mark M), and prepares a tuple with the answered tests signed by the authority and the marking, which E signs further with its secret key Sk_E. Encrypted this with the public key of the authority, E returns this message to the A who, in turn, decrypts the marking and re-encrypts it with the pseudonym of the candidate Pk_C.

At *notification* the candidates' anonymity is revoked and the marks can finally be registered. The phase starts when the authority publishes all the marks together. On request the mixnets reveal (on an authenticated encrypted channel like one using TLS) the random values used to generate the pseudonyms (in Figure, $\overline{r}_m$, for C).

Security properties Remark! ensures several authentication, anonymity, privacy and verifiability properties. Here, we focus on verifiability, in particular on *universal verifiability*. Universal means that anyone, even with no knowledge about the protocol execution, can verify that a specific property holds over the execution. A universal verifiability test should use only the public data available on the bulletin board, plus pieces of additional information which are explicitly provided for the test.

Remark! has been proven to be verifiable for five properties (Dreier et al., 2014):

***Registration*:** all accepted tests are submitted by registered candidates.

***Marking Correctness*:** all the marks attributed by the examiners to the tests are computed correctly.

***Test Integrity*:** all and only accepted tests are marked without any modification

***Test Markedness*:** only the accepted tests are marked without modification

***Mark Integrity*:** all and only the marks associated to the tests are assigned to the corresponding candidates with no modifications

The properties are verifiable only if the exam authority handles to the verifier some data *after* the exam has ended. For instance to verify *Registration* the manager must reveal the signatures inside the receipts $\{Sign_{\mathsf{Sk}_A}(H(T_C)\}_{\overline{\mathsf{Pk}}_C}$ posted on the bulletin board and the random values used to encrypt the receipts. To verify *Marking Correctness*, the manager must reveal the marked exam-tests inside the evaluations $\{Sign_{\mathsf{Sk}_E,h_E}(M_C)\}_{\mathsf{Pk}_A}$, the random values used to encrypt the marked tests, and the table *correct_ans* (not described in Figure 1) that defines what is the mark given a question and an answer; to verify *Test Integrity* and *Test Markedness*, the manager must reveal the marked exam-tests inside the evaluations, the random values used to encrypt the marked tests, plus the data disclosed for Registration; finally, to verify *Mark Integrity*, the manager must reveal the examiners' signatures on the marked exam-tests inside the evaluations, and the random values used to encrypt the notifications $\{Sign_{\mathsf{Sk}_E,h_E}(M_C)\}_{\overline{\mathsf{Pk}}_C}$ before posting them on the bulletin board.

Discussion Let us see where Remark!'s verifiability fails to be privacy-preserving.

To verify *Registration*, Remark! prescribes a test that takes the pseudonyms of the candidates signed by the mixnet, i.e., message $Sign_{SK_M}(\overline{\mathsf{Pk}}_C, h_C)$, and the receipts of submissions generated by the exam authority i.e., messages $\{Sign_{SK_A}(H(T_C)\}_{\overline{\mathsf{Pk}}_C}$. The test succeeds if

for each pseudonym there is a unique receipt of submission. The verifier does not know any other information about the execution of the exam apart the pseudonyms and the hash version of the tests. Hence, the registration test satisfies Definition 4.

To verify *Marking Correctness*, the protocol requires a test that inputs $Sign_{SK_M}(\overline{\mathsf{Pk}}_C, h_C)$, the table *correct_ans*, and the mark notifications signed by the examiner and published by the exam authority $\{Sign_{SK_E,h_E}(M_C)\}_{\mathsf{Pk}_A}$. The test needs also the help of the exam authority that decrypt these last messages, and reveals the $M_C = \langle Sign_{SK_A}(\overline{\mathsf{Pk}}_E, T_C), mark\rangle$ which reveals the pseudonym of the examiner who marked the answers, the questions, and the mark. This test does not satisfy Definition 4: the auditor learns the questions, the answers, and the link between candidate and examiner pseudonyms.

To verify *Test Integrity*, the protocol needs a test that takes in $Sign_{SK_M}(\overline{\mathsf{Pk}}_C, h_C)$, the receipts of submissions $\{Sign_{SK_A}(H(T_C)\}_{\overline{\mathsf{Pk}}_C}$, and the mark notifications generated by the exam authority in $\{Sign_{SK_E,h_E}(M_C)\}_{\overline{\mathsf{Pk}}_C}$. This test reveals to the auditor questions, answers, and the link between candidate and examiner pseudonyms, which are leaked in the mark notifications, so it does not satisfy Definition 4

To verify *Test Markedness* a test needs to input the same data as in test integrity. The same considerations outlined above applies for this test and the test does not satisfy Definition 4.

To verify *Mark Integrity*, Remark! prescribes a test that takes in the mark notifications $\{Sign_{SK_E,h_E}(M_C)\}_{\overline{\mathsf{Pk}}_C}$ and allows the auditor to check if the marks that the exam authority assigned to the candidates (Register $\mathsf{Pk}_C, mark$) coincide with the marks that the examiner assigned to the candidates tests in $M_C = \langle Sign_{SK_A}(\overline{\mathsf{Pk}}_E, T_C), mark\rangle$. Once more, the knowledge of questions, answers, and link between candidate and examiner pseudonyms is leaked by M_C, although those pieces of information are not necessary to evaluate mark integrity. Also this test does not respect Definition 4.

Overall, Remark! is privacy-preserving only for registration verifiability.

5 Applicability

Remark! offers only one test of verifiability that is privacy-preserving, *Registration*. It fails to offer the same for the other tests of universal verifiability. Here, we show how to make them privacy-preserving. For reason of space we show the technique only on the universal version of Mark and Test Integrity Verifiability. We claim, but we have to leave the proof of this claim as future work, that we can apply the same idea of solution to achieve privacy-preserving verifiability for all the other verifiability tests.

To achieve privacy-preserving verifiability for Mark and Test Integrity verifiability we modify Remark!. We propose a version that relies on Functional Encryption. We recall the basic of this theory in § 5.1. We give the new protocol's specification in § 5.2.

5.1 Functional Encryption

Functional Encryption (FE) is an encrypting paradigm. It relies on the notion of FE scheme (Boneh et al., 2011).

Definition 6. *A* FE scheme *is a 4-tuple of functions* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval})$*, where:*

1. $\mathsf{Setup}(1^\lambda, 1^n)$ *returns a pair of keys* $(\mathsf{Pk}, \mathsf{Msk})$*. The first is a public key and the other a secret master key. The keys depends on two parameters:* λ*, a security parameter and* n*, a length. They are polynomially related.*
2. $\mathsf{KeyGen}(\mathsf{Msk}, C)$ *is a function that returns a* token, $\mathtt{Tok}_C^{\mathsf{Msk}}$*. A token is a string that when evaluated (see item 4, function* Eval*) on specific ciphertexts encrypted with* Pk *returns the same output as the* Boolean Circuit C *(a decider) would return if it were applied on the corresponding plaintexts. We talk about Boolean Circuits (Jukna, 2012) because the "functionality" in "functional encryption" is defined on this model of computation. Here,* $C \in \mathcal{C}_n$*, wheren* $\mathcal{C}_n$ *is the class of Boolean Circuit on inputs long* n *bits.*
3. $\mathsf{Enc}(\mathsf{Pk}, m)$ *encrypts the plaintext* $m \in \{0,1\}^n$ *with the public key* Pk*.*
4. $\mathsf{Eval}(\mathsf{Pk}, \mathsf{Ct}, \mathtt{Tok}_C^{\mathsf{Msk}})$ *is the function that processes the token as we anticipated in item 2. On input* $\mathsf{Ct} = \mathsf{Enc}(\mathsf{Pk}, m)$ *the function returns* $C(m)$ *that is the same boolean output (accept/reject) as that of circuit* C *applied on string* m*. The key* Msk *embedded in the token is used retrieve the plaintext* m *from* Ct*. For any other input than the specified* Ct*,* Eval *is undefined and returns* $\perp$*.*

A FE scheme should be of course instantiated cryptographically, but whatever implementation

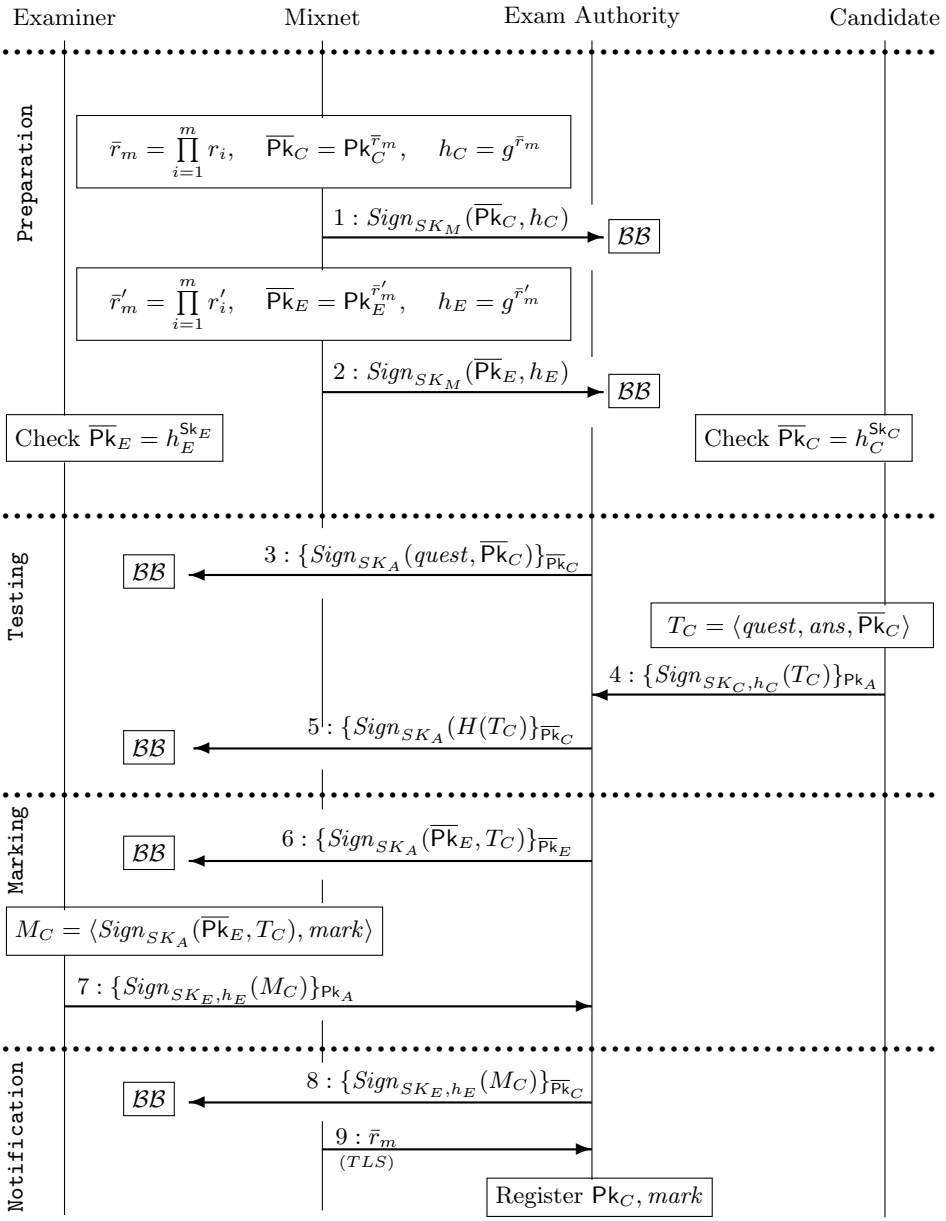

Figure 1: The message flow of Remark!

we chose it must satisfy two specific requirements: *correctness* and *secure indistinguishability*.

The requirement of *correctness* says that the functions in Definition 6 behave as described: for all $(\mathsf{Pk}, \mathsf{Msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, and all $C \in \mathcal{C}_n$ and $m \in \{0,1\}^n$, and for $\mathtt{Tok}_C^{\mathsf{Msk}} \leftarrow \mathsf{KeyGen}(\mathsf{Msk}, C)$ and $\mathsf{Ct} \leftarrow \mathsf{Enc}(\mathsf{Pk}, m)$ then $\mathsf{Eval}(\mathsf{Pk}, \mathsf{Ct}, \mathtt{Tok}_C^{\mathsf{Msk}}) = C(m)$. Otherwise it returns $\perp$.

The requirement of *secure indistinguishability* says that the scheme is "cryptographically secure" against any active and strong enough adversary (i.e., uniform probabilistic polynomial time adversaries). Such an adversary, despite having seen tokens $\mathtt{Tok}_C^{\mathsf{Msk}}$, cannot distinguish (with a non-negligible advantage over pure guessing) two ciphertexts $\mathsf{Enc}(\mathsf{Pk}, m_0)$ and $\mathsf{Enc}(\mathsf{Pk}, m_1)$ for which $C(m_0) = C(m_1)$.

5.2 Revising the Protocol

We modify Remark! and let it have a test for *Universal Marking and Test Integrity* (UMTI) verifiability that is privacy-preserving according to our Definition 4. We use a FE scheme to encrypt the messages that the verifier needs to process when running the test. From §3 we recall that the test should decide the following predicate:

$$\begin{aligned}\mathtt{test}_{UMTI}(\{\mathcal{E}\}, K) \text{ iff } (\mathtt{NotifiedMarks} \subseteq \\ \{(c,m) : (c,(q,a),m) \in \mathtt{MarkedTests} \\ \text{and } (c,(q,a)) \in \mathtt{SubmittedTests}\})\end{aligned} \quad (4)$$

The test should succeed if and only if the set of all notified marks to candidates is included in the set of exams that have been marked for those candidates and the exams marked should be those that the have submitted for marking.

As we discussed in §4, in Remark! this test is implemented by letting the verifier all the messages $Sign_{SK_E,h_E}(M_C)$ from which he can build the sets as in Equation (4). By doing so the verifier gets to know the questions and answers and their link with the candidate identifiers.

By resorting to a FE scheme we intend to achieve a version of the test that implements predicate (4) but without revealing the candidate's questions and answers, which is a sufficient condition to satisfy Definition 5. Precisely, Remark! is modified as in Figure 2, showing only the part of the protocol that has been modified.

We assume a trusted authority which we call *FE authority*. It is not shown in Figure 2. Messages in step 5 and in step 8, in red in Figure 2, are those which change. They are encrypted with the FE's public key. Msk. The authority can be any party, such as one participating in the protocol or an external one, but must be trusted in this particular task.

Precisely, in step 5 the exam authority encrypts with $\mathsf{Pk_{FE}}$ the signature of a test T_C that the candidate submitted for marking. It also posts a digest of the same message, signed with the candidate's pseudonym. To avoid that the message encrypted under $\mathsf{Pk_{FE}}$ and that encrypted under Pk_C hide two different values (i.e., y and T_C such that $H(T_C) \neq y$) we require a non-interactive zero knowledge proof (NIZK) of consistency, π in Figure. Similarly, in step 8 the authority also encrypts with $\mathsf{Pk_{FE}}$ the marking it has received from the examiner. Another NIZK π' is required as a proof of consistency.

5.3 Revising the Verifiability Test

Ciphertexts in steps 5 and in 8 will be used in combination with the tokens that the FE authority prepares for the verifier. The verifier needs two type of tokens: they "implement" two functions.

The first, function f_1, embeds the public-keys $\overline{\mathsf{Pk}}_C$ and Pk_E and the mark *mark* to be verified. It is supposed to input $Sign_{\mathsf{Sk}_E,h_E}(M_C)$, the message in 8. After properly parsing the strings, checks whether the mark, say *mark′*, contained in M_C is equal to *mark*, checks the signature and checks whether the $\overline{\mathsf{Pk}}'_C$ contained in T_C (in turn, contained in M_C) equals $\overline{\mathsf{Pk}}_C$. It also verifies the validity of the zero-knowledge proof π. If a check fails, the function returns 0. If all checks succeed, it returns 1. Function f_1 is coded as follows:

Function $f_1[\overline{\mathsf{Pk}}_C, mark, \mathsf{Pk}_E](\sigma)$
1. If $\textsc{Verify}(\sigma, \mathsf{Pk}_E) = 0$ then Return 0
//verifies a signature, returns 0 on fails
2. Parse σ as $Sign_{\mathsf{Sk}_E,h_E}(M_C)$
3. Parse M_C as $\langle Sign_{SK_A}(\overline{\mathsf{Pk}}_E, T_C), mark' \rangle$ (get $mark', T_C$)
4. Parse T_C as $\langle quest, ans, \overline{\mathsf{Pk}}'_C \rangle$ (get Pk'_C)
5. If $mark' \neq mark$ Return 0
6. If $\overline{\mathsf{Pk}}'_C \neq \overline{\mathsf{Pk}}_C$ Return 0
7. Return 1

The second function, f_2, embeds the public-key $\overline{\mathsf{Pk}}_C$ and the message $Sign_{\mathsf{Sk}_E,h_E}(M_C)$. It expects to input the second part of message 5, $Sign_{\mathsf{Sk}_C,h_C}(T_C)$. It checks the signatures and whether both messages refer to the same exam test. The function returns 0 as soon as one check fails, 1 otherwise. Function f_2 is coded as follows:

Function $f_2[\overline{\mathsf{Pk}}_C, Sign_{\mathsf{Sk}_E,h_E}(M_C)](\sigma)$
1. If $\textsc{Verify}(\sigma, \mathsf{Pk}_C) = 0$ then Return 0
//verifies a signature, returns 0 on fails
2. Parse σ as $Sign_{\mathsf{Sk}_C,h_C}(T_C)$ (get T_C)
3. Parse M_C as $\langle Sign_{SK_A}(\overline{\mathsf{Pk}}_E, T'_C), mark \rangle$ (get T'_C)
4. If $T_C \neq T'_C$ Return 0
5. Return 1

The verifiability test uses two sets of tokens of form $\mathtt{Tok}^{\mathsf{Msk}}_{f_1[x]}$ and $\mathtt{Tok}^{\mathsf{Msk}}_{f_2[x]}$ where x ranges over the token embedded values.

The test then calls the function Eval over those tokens to check whether the f_1 and f_2 holds on the messages found on the bulletin board that correspond the the messages that the verifier needs to implement the test in Equation (4).

$\mathtt{test}_{UMTI}(\{\mathcal{E}\}, K)$ first verifies whether for each notified mark per candidate there exists at least one message among those in step 8 that corresponds to the mark for a test of that candidate; if one is found the test continues and checks whether there is at least one message among those in step 5 that corresponds to that exam for that candidate submitted for marking.

The test requires to know the markings that have been notified to the candidates and PK_E the public key of the authority, so $K = \{(\mathsf{Pk}_C, mark_C) : C \in \mathsf{ID}_{rc}\} \cup \{\mathsf{Pk}_E\}$. Formally the test is implemented by the following algorithm:

function $\mathtt{test}_{UMTI}(\{\mathcal{E}\}, K)$:

for $(\mathsf{Pk}_C, mark_C) \in K$ **do**
$\mathtt{Tok}_1 \leftarrow$ GETTOK$(f_1[\mathsf{Pk}_C, mark_C, \mathsf{Pk}_E])$
for $(\{Sign_{\mathsf{Sk}_E,h_E}(M_C)\}_{\mathsf{Pk_{FE}}}, \pi) \in \{\mathcal{E}\}$ **do**

$b_1 \leftarrow \mathsf{Eval}(\mathtt{Tok}_1, \{Sign_{\mathsf{Sk}_E,h_E}(M_C)\}_{\mathsf{Pk_{FE}}})$;
if b_1 **then** ExitLoop;
if b_1 **then**
$b'_1 \leftarrow$ VERIFY(π for R);
$\mathtt{Tok}_2 \leftarrow$ GETTOK$(f_2[Sign_{\mathsf{Sk}_E,h_E}(M_C)])$;
for $(\{Sign_{\mathsf{Sk}_C,h_C}(T_C)\}_{\mathsf{Pk_{FE}}}, \pi') \in \{\mathcal{E}\}$ **do**

$b_2 \leftarrow \mathsf{Eval}(\mathtt{Tok}_2, \{Sign_{\mathsf{Sk}_C,h_C}(T_C))\}_{\mathsf{Pk_{FE}}})$;
if b_2 **then** ExitLoop
if b_2 **then**
$b'_2 \leftarrow$ VERIFY(π' for R');
return $b_1 \cdot b'_1 \cdot b_2 \cdot b'_2$

GETTOK is used to retrieve a specific token. Here, we use the function's embedded parameters c to indicate which token is being requested. However, the verifier does not necessarily know c to perform such request, but he must be able to indicate the token he needs e.g., by means of an index. Relations R and R' are so defined:

Relation $\mathsf{R}[\overline{\mathsf{Pk}_C}, \mathsf{Pk_{FE}}, H](x)$
1. Parse x as
$(\mathsf{Ct}_1 = \{Sign_{SK_A}(y)\}_{\overline{\mathsf{Pk}}_C}$,
$\mathsf{Ct}_2 = \{Sign_{\mathsf{Sk}_E,h_E}(T_C)\}_{\mathsf{Pk_{FE}}})$
(get y and T_C)
2. If $H(T_C) \neq y$ Return 0
3. Return 1

and

Relation $\mathsf{R}'[\overline{\overline{\mathsf{Pk}_C}}, \mathsf{Pk_{FE}}](x)$
1. Parse x as
$(\mathsf{Ct}_1 = \{Sign_{SK_E,h_E}(y)\}_{\overline{\mathsf{Pk}}_C}$,
$\mathsf{Ct}_2 = \{Sign_{\mathsf{Sk}_E,h_E}(T_C)\}_{\mathsf{Pk_{FE}}})$
(get y and T_C)
2. If $T_C \neq y$ Return 0
3. Return 1

5.4 Discussion

We argue that $\mathtt{test}_{UMTI}(\{\mathcal{E}\}, K)$ is privacy preserving. By assumption the FE scheme satisfies the indistinguishability security. Thus, a Dolev-Yao attacker, given a ciphertext Ct encrypting a message m and a token for a function f, can only derive $f(m)$. This holds also for a curious verifier who observes the tokens for the two functions f_1 and f_2. The crucial consideration is that, by construction, such two functions return a *Boolean* value and nothing about the messages that are considered in our model (cf. Def. 1) of an exam's execution. Moreover, among the messages that the verifier gets in input, precisely in K there is nothing that allows him to decrypt other messages on the bulletin board. Actually, all the decryption operations that the verifier needs are realized within the function Eval. Therefore, it follows that $\mathtt{test}_{UMTI}(\{\mathcal{E}\}, K)$ is privacy preserving according to Definition 4.

Name	Equation
Signature	$getmess(sig(m,k)) = m$ $checksig(sig(m,k), spk(k)) = m$ $checksig(sig(m, ps\_pr(k,$ $exp(rce))), ps\_pub(pk(k), rce)) = m$
ElGamal	$decrypt(enc(m, pk(k), r), k) = m$ $decrypt(enc(m, ps\_pub(pk(k),$ $rce), r), ps\_pr(k, exp(rce))) = m$ $checkps(ps\_pub(pk(k), rce),$ $ps\_pr(k, exp(rce))) = true$
FE f1	$eval\_f1(pk(k), ps\_pub(pk(k), rc),$ $ps\_pub(pk(k), re), mark,$ $enc(sig(sig(ps\_pub(pk(k), re),$ $(q, a, ps\_pub(pk(k), rc)), k),$ $mark, ps\_pr(k, exp(re))), pk(k))) = true$
FE f2	$eval\_f2(pk(k), ps\_pub(pk(k), rc),$ $enc(sig((q, a, ps\_pub(pk(k), rc)),$ $ps\_pr(k, exp(rce))), pk(k))) = true$

Table 1: Equational theory to model Remark!

6 Formal Analysis

We prove formally that the verifiability test is sound and complete and that the new protocol does satisfy the same security properties as the original Remark!. That is because we intend to get insurance than we achieve privacy-preserving verifiability incrementally, on top of all the authentication, integrity and privacy properties holding on the protocol and not of their expenses.

We perform the analysis in Proverif (Blanchet, 2014). Our ProVerif model of the protocol is an extension of the original Remark! model as it was presented in (Dreier et al., 2014). The extension mainly regards the following aspects:

- A ProVerif specification that reflects the new protocol description.
- A new equational theory for the functional encryption primitives defined in our protocol.
- A new UMTI verifiability test.

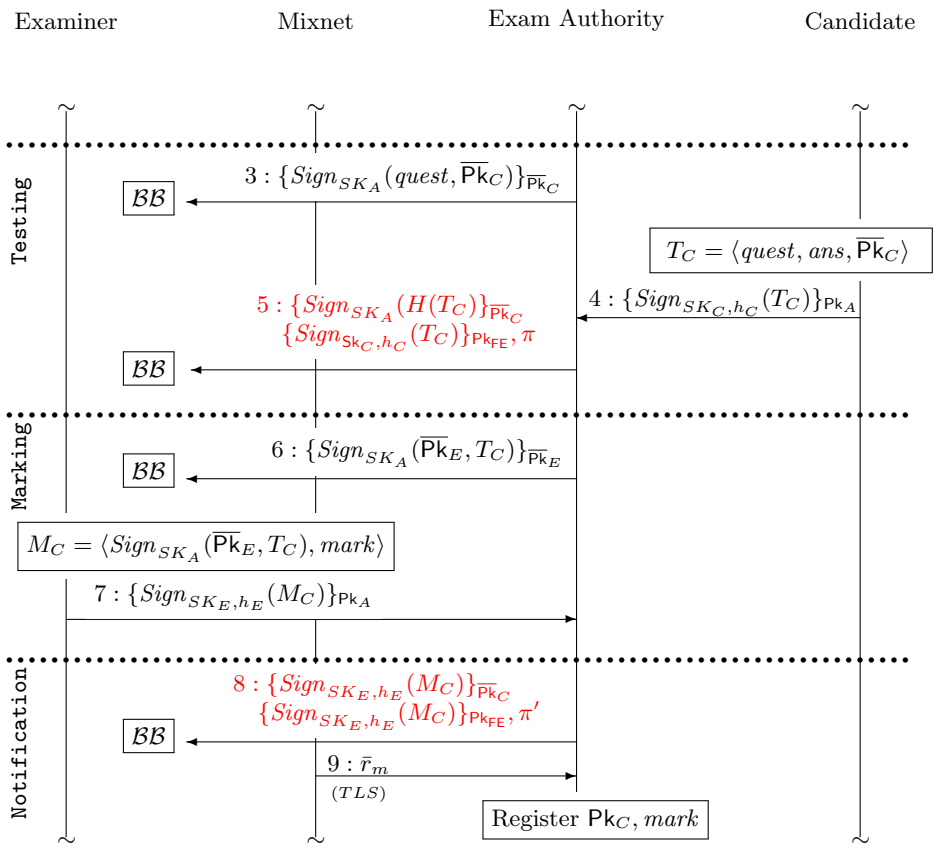

Figure 2: A revision of Remark! using FE (only phases that have been changed).

The equational theory is illustrated in Table 1 to model the cryptographic primitives of the protocol. The equations that model the digital signature are rather standard in ProVerif.

ElGamal encryption is extended with equations that model pseudonyms as public keys: the pseudonym, which also serves as test identifier, can be generated using the function *ps_pub* which takes in a public key and a random exponent. In fact, this function models the main feature of exponentiation mixnet. Function *ps_pr* can be used by a principal to decrypt or sign anonymous messages. The function takes in the private key of the principal and the new generator published by the mixnet. Function *checkps* allows a principal to check whether a pseudonym is associated with the principal's private key. In practice, principals use this function to identify their pseudonyms published on the bulletin board. A public channel models the bulletin board.

The equations that defines *eval_f1* and *eval_f2* model the two functional encryption primitives f_1 and f_2 respectively. They return *true* if and only if the corresponding plaintext version f_1 and f_2 return *true*. Note that our equational theory of functional encryption captures the property of verifiable FE. Whenever the ciphertext is not in the range of the encryption (i.e., it is not consistent with the constructor *enc*), the evaluation function *eval* fails as no rule can be applied. We do not explicitly model the generation of the token: Setup and KeyGen are generated by a trustworthy authority. It follows that the only two allowable functions are f_1 and f_2.

Verifiability. Recalling from Definition 3, a protocol is p verifiable if it admits an algorithm that decides p and the algorithm is sound and complete. Soundness means that the verifiability-test returns `true` only if the property holds. Completeness means that if the property holds, the verifiability-test always returns `true`.

We use ProVerif's correspondence assertions to prove soundness. The verification strategy consists of checking that the event `OK` emitted by the verifiability-test when it is supposed to return `true` is always preceded by the event emitted in the part of the code where the property becomes satisfied. In case of Mark and Test Integrity, this happens when the exam authority assigns the mark to the candidate.

We resort to the unreachability of the event `KO`, which is emitted by the verifiability-test when it is supposed to return `false`, to prove complete-

ness. In this case, the ProVerif model enforces only honest principals and prevents the attacker to manipulate the input data of the verifiability-test. In fact, a complete verifiability-test must succeed if its input data is correct. ProVerif confirms that our protocol meets UMTI verifiability and the verifiability-test is sound and complete [1].

We observe that ProVerif can only prove the case in which only one entry on the bulletin board is considered. Mark Integrity verifiability requires that *all* and only the marks associated to the tests are assigned to the corresponding candidates. Thus, we need to iterate over all candidates who submitted their tests, but ProVerif cannot do this automatically as it does not support loops. We resort on the manual induction proofs provided by Dreier et al. (Dreier et al., 2014) in which the base case is given by the automated result in ProVerif, and then generalise such result to the general case with an arbitrary number of candidates, tests, and marks. Those proofs applies to the original Remark! as well as to this new version. In fact, the assumption that the auditor can check the correspondences of the assigned entries to the marked entries on the bulletin board does not change: each entry that appears on the bulletin board at step 6 of the protocol can be matched to an entry that appears on the bulletin board at step 8 of the protocol.

Authentication. We also prove in ProVerif that the new protocol meets all the original authentication properties of Remark! Since the new protocol differs from Remark! only in messages 5 and 8, we can reuse the same authentication definitions advanced by Dreier et al. (Dreier et al., 2014).

7 Discussion and Conclusion

In this pioneering work, we put forth a new notion of security that entails features of both verifiability and privacy. To illustrate the benefits of our framework, we proved that our definition is meaningful enough to empower applications in the concrete domain of electronic exams and in particular to an extension of an exam protocol that satisfies several verifiability properties but which is not compliantly with our privacy-preserving requirement.

[1]The full ProVerif code is available for review via a request to the PC chairs

To achieve our goals, we employ the cryptographic tool of Functional Encryption (FE). Only with FE we can verify obliviously a property that is global over the execution of the protocol, a property that is about the data handled and processed by several agents in the protocol's run. In making this choice we considered to obtained the same effect with other tools, the most promising be zero knowledge proofs, but, we claim, using this tool would have required an authority that looks at the execution and adds the proofs and that interacts with the several protocol agent's asking them for other zero-knowledge proofs, so changing too much how an exam should work. This argument requires however a proof and we plan to inquire into this question as future work.

We have not benchmarked the new protocol, since this work's focus is supposed to be mainly theoretical. At current stage, our result is only of theoretical relevance due to the high computational cost of FE for circuits that we assumed and for this reason we recognize that functional encryption may look as be an overkilling to someone. However, there is another feature of electronic exams that must be considered here. The limited dimension of the an exam's audiences and the expected time of an exam's notification makes should make feasible implementations that rely on time-inefficient encryption schemes. Comparing with electronic voting, for instance, where a whole country is involved and where a result is nowadays expected be announced within the day, for an exam the expected audiences is definitely far more contained while waiting weeks is a perfectly acceptable time frame to get notified of the result. We defer to further research about implementing our privacy-preserving verifiability notion efficiently.

We conclude by pointing to a new potential interesting research direction, a future work for us, and open problem to whom it may be interested: to study the relation between our notion and that presented by Mödersheim *et al.* of α-β privacy (Mödersheim et al., 2013). Were this correlation proved, we could gain a straightforward way to verify formally privacy-preserving verifiability through the fact α-β privacy subsumes static equivalence.

REFERENCES

Adida, B. and Neff, C. A. (2006). Ballot Casting Assurance. In *Proc. of the*

USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop, EVT'06, pages 7–7. USENIX Association.

Benaloh, J. (1996). *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University.

Benaloh, J. and Tuinstra, D. (1994). Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the 26th Symposium on Theory of Computing (STOC '94)*, pages 544–553, New York, NY, USA. ACM.

Blanchet, B. (2014). Automatic Verification of Security Protocols in the Symbolic Model: the Verifier ProVerif. In *Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures*, volume 8604 of *LNCS*, pages 54–87. Springer.

Boneh, D., Sahai, A., and Waters, B. (2011). Functional encryption: Definitions and challenges. In Ishai, Y., editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273, Providence, RI, USA. Springer, Berlin, Germany.

Cohen, J. and Fischer, M. (1985). A robust and verifiable cryptographically secure election scheme (extended abstract). In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 372–382, Portland, Oregon, USA. IEEE Computer Society.

Cuvelier, E., Pereira, O., and Peters, T. (2013). *Election Verifiability or Ballot Privacy: Do We Need to Choose?*, pages 481–498. Springer Berlin Heidelberg, Berlin, Heidelberg.

De Santis, A., Micali, S., and Persiano, G. (1988). Non-interactive zero-knowledge proof systems. In Pomerance, C., editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 52–72, Santa Barbara, CA, USA. Springer, Berlin, Germany.

Department of Defence (1987). DoD Personnel Security Program. DOD 5200.2 R.

Dreier, J., Giustolisi, R., Kassem, A., Lafourcade, P., and Lenzini, G. (2015). A Framework for Analyzing Verifiability in Traditional and Electronic Exams. In *Proc of the 11th Int. Conf. Information Security Practice and Experience (ISPEC 2015), Beijing, China, May 5-8, 2015*, pages —.

Dreier, J., Giustolisi, R., Kassem, A., Lafourcade, P., Lenzini, G., and Ryan, P. Y. A. (2014). Formal analysis of electronic exams. In *SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, August*, pages 101–112. SciTePress.

Giustolisi, R., Lenzini, G., and Ryan, P. Y. A. (2014). *Remark!: A Secure Protocol for Remote Exams*, pages 38–48. Springer International Publishing, Cham.

Hirt, M. and Sako, K. (2000). Efficient receipt-free voting based on homomorphic encryption. In *Proceedings of the 19th Annual Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT'00)*, volume 1807 of *LNCS*, pages 539–556. Springer.

Jukna, S. (2012). *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer.

Kremer, S., Ryan, M., and Smyth, B. (2010). Election verifiability in electronic voting protocols. In *Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS'10)*, volume 6345 of *LNCS*, pages 389–404. Springer.

Küsters, R., Truderung, T., and Vogt, A. (2010). Accountability: definition and relationship to verifiability. In *Proceedings of the 17th Conference on Computer and Communications Security (CCS'10)*, pages 526–535. ACM.

Mödersheim, S. A., Groß, T., and Viganò, L. (2013). *Defining Privacy Is Supposed to Be Easy*, pages 619–635. Springer Berlin Heidelberg, Berlin, Heidelberg.

Naveed, M., Kamara, S., and Wright, C. V. (2015). Inference Attacks on Property-Preserving Encrypted Databases. In *Proc. of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 644–655, New York, NY, USA. ACM.

Smyth, B., Ryan, M., Kremer, S., and Mounira, K. (2010). Towards automatic analysis of election verifiability properties. In *Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10)*, volume 6186 of *LNCS*, pages 146–163. Springer.