Automated Analysis of Accountability

Alessandro Bruni, Rosario Giustolisi(✉), and Carsten Schuermann

IT University of Copenhagen, Copenhagen, Denmark
`rosg@itu.dk`

Abstract. A recent trend in the construction of security protocols such as voting and certificate management systems is to make principals *accountable* for their actions. Whenever some principals deviate from the protocol's prescription and cause the failure of a goal of the system, accountability ensures that the system can detect the misbehaving parties who caused that failure. Accountability is an intuitively stronger property than *verifiability* as the latter only rests on the possibility of detecting the failure of a goal. A plethora of accountability and verifiability definitions have been proposed in the literature. Those definitions are either very specific to the protocols in question, hence not applicable in other scenarios, or too general and widely applicable but requiring complicated and hard to follow manual proofs.

In this paper, we advance formal definitions of verifiability and accountability that are amenable to automated verification. Our definitions are general enough to be applied to different classes of protocols and different automated security verification tools. Furthermore, we point out formally the relation between verifiability and accountability. We validate our definitions with the automatic verification of three protocols: a secure exam protocol, Google's Certificate Transparency, and an improved version of Bingo Voting. We find through automated verification that all three protocols satisfy verifiability while only the first two protocols meet accountability.

1 Introduction

In the real world, disputes among principals can be resolved with trials. A judge or jury will decide on a trial according to the evidence presented by the parties. In the digital world, even if the design of a security protocol is sound, dishonest principals may still attempt attacks that cause protocol functional failures. Similarly to real-world protocols, principals should be able to raise disputes in which a judge blames principals who caused the failure according to the evidence. This notion is known as *accountability* and ensures that (i) failures are detectable and (ii) misbehaving principals can be blamed. Accountability is a stronger notion than *verifiability* as the latter only requires that the failure of a protocol's goal can be detectable [1]. Thus, security protocols should be designed to provide adequate evidence to enable accountability. In so doing, principals are discouraged to misbehave, fostering minimal intentional protocol failures.

© Springer International Publishing AG 2017

Contribution. The goal of this paper is to fully mechanise the analysis of verifiability and accountability in security protocols. We propose definitions based on the existence of an accountability test that decides whether a principal should be blamed for the failure of a protocol's goal. We conveniently adapt a generic definition of protocol advanced by Küsters et al. [2] to specify the soundness and completeness conditions for accountability tests that can be checked by automated security protocol tools. We show that verifiability is a necessary condition for accountability and our treatment of accountability is general enough to apply to different tools and protocols. Then, we validate our definitions in three different case studies with two different tools. The first case study is about a secure exam protocol, and we check accountability with ProVerif [3]. The second case study concerns Google's Certificate Transparency, and we prove accountability with AIF-ω [4]. The third case study considers an improved version of Bingo Voting, which is analysed again with ProVerif.

Outline. The paper is organised as follows. Section 2 discusses some related work. Section 3 details our definitions of verifiability and accountability. Section 4 validates the definitions in a secure exam protocol. Section 5 details the formal analysis of Google's Certificate Transparency. Section 6 analyses verifiability and accountability in Bingo Voting. Finally, Sect. 7 concludes the paper.

2 Related Work

In this paper we define an accountability test, which can be used to decide if a protocol is accountable, meaning if it has the capability to single out reliably the parties (if any) that are compromised and behaving dishonestly. A precondition for our accountability test is verifiability that is designed to detect if something went wrong in the first place. The hallmark characteristic of our accountability and verifiability definitions is that they are mechanizable in the symbolic model. The definition of our criterion is formalism and tool independent, which sets it apart from related projects, which we discuss briefly below.

Our work builds on the work by Küsters et al. [2] who define notions of accountability and verifiability in the symbolic and computational models. The symbolic definitions aim at precisely describing the assessment of the level of accountability that a protocol provides. This comes at the cost of definitions that may not be amenable to automated analysis as the verification approach would heavily depend on the accountability property under consideration. Differently, our definitions are explicitly adapted for checking accountability with automated security protocol tools. To aid the reader familiar with Küsters et al. work in comparing our work to theirs, we revisit in this paper the example of Bingo Voting, whose analysis was supported with manual proofs in Küsters et al.'s work.

Milner et al. [5] focus on a provably sound detection of misuse of secrets. Their work has yielded new insights into detecting the misuse of a Certification Authorities key on the Internet and contributed to the broader area of Certificate

Transparency. Again, to aid the reader familiar with this work to compare their results to ours, we demonstrate how to mechanise this argument in AIF-ω [4].

Jagadessan et al. [6] proposed a framework that deals with the general notion of accountability but cannot deal with cryptography. Bella and Paulson [7] advanced a computer-assisted analysis of accountability of the Zhou-Gollman non-repudiation protocol [8]. Similarly, Abadi and Blanchet [9] analysed accountability for a certified email protocol. The definitions proposed in these works are not general but specific to the protocols in question.

The notion of verifiability has been extensively studied in voting [1,10]. The notion of *individual verifiability* signifies that voters can verify that their votes have been handled correctly, namely "cast as intended", "recorded as cast", and "counted as recorded" [11,12]. The notion of *universal verifiability* has been introduced to express the concept in which auditors can verify the correctness of the tally using only public information [11,13,14]. Kremer et al. [10] formalised both individual and universal verifiability in the applied pi-calculus. They also introduced the requirement of *eligibility verifiability*, which expresses that auditors can verify that each vote in the election result was cast by a registered voter, and there is at most one vote per voter. Smyth et al. [15] used ProVerif to check verifiability in three voting protocols. They express the requirements as reachability properties. Similarly, Dreier et al. [16] checked in ProVerif soundness and completeness conditions for verifiability-tests in three auction protocols. In this paper, we also analyse two security protocols in ProVerif. However, our definitions of verifiability and accountability are constrained neither to the applied pi-calculus nor ProVerif.

Guts et al. [17] defined *auditability* as the quality of a protocol, which stores a sufficient number of pieces of evidence, to convince an honest judge that specific properties are satisfied. Auditability is a weaker notion of accountability and expresses the same concept of universal verifiability: anyone, even an outsider without a private knowledge about the protocol execution, can verify that the system relies only on the available pieces of evidence.

3 Definitions

We begin our formal treatment with the formal definition of a protocol, following roughly the exposition of Küsters et al. [2]. Our definitions differ from theirs to support better the mechanisation effort discussed below.

Definition 1 (Protocol). *A* protocol *is a tuple* $P = \langle \mathsf{Ch}, \mathsf{A}, \Pi, \mathsf{G} \rangle$ *such that:*

- $\mathsf{Ch} = \{ch_1, \ldots, ch_n\}$ *is a set of* channels*;*
- $\mathsf{A} = \{\alpha_1, \ldots, \alpha_n\}$ *is a set of* principals*;*
- Π *is the set of* programs *run by the principals;*
- G *is the set of* goals *that the protocol aims to meet.*

Given a set of primitive operations, for example, for sending and receiving messages on channels, encrypting and decrypting messages using keys, etc. we refer to sequences of such operations as *programs*. The set of all such programs is denoted by Π, with the intention that for each run of the protocol each principal $\alpha_i \in \mathsf{A}$ is expected to running one and only one such program $\pi_{\alpha_i} \in \Pi$. We write r for a *run* of the protocol. Each run produces a trace. A *witness trace*, which we denote with t, is a run of the protocol from the point of the view of a principal and serves as input and evidence for the verifiability and accountability tests. We do not distinguish between input and output channels. Instead, we introduce predicates $g \in G$ that range over traces and distinguish the traces that achieve the *goal* of a protocol from those that do not. As we shall see later, verifiability and accountability definitions are pivoted on protocol's goal. Thus, we detail its treatment here to obtain clearer definitions later.

For each set of goals G, we define Π^G as the set of all tuples $\{\pi_{\alpha_i}\}_{\alpha_i \in \mathsf{A}}$, where each such tuple defines one program for each respective principal, that converge towards satisfying all the goal defined within G, when running in parallel, as $(\pi_{\alpha_1}|\pi_{\alpha_2}|\dots|\pi_{\alpha_n})$. For instance, let us consider two principals, Alice and Bob, who will communicate over some channel. The goal g of this protocol is that Bob eventually receives some message. Let us consider the protocol consisting of two programs $(\pi^1_{\text{Alice}}, \pi^1_{\text{Bob}})$ that Alice and Bob are expected to run. The first program consists of π^1_{Alice} that sends one message while the other π^1_{Bob} expects to receive some message. Now, let us assume that Alice runs a different program π^2_{Alice} that sends two messages. Although Alice runs a program that deviates from the original protocol prescription, the tuple of programs $(\pi^2_{\text{Alice}}, \pi^1_{\text{Bob}})$ still clearly converges towards the goal. Consequently, $(\pi^1_{\text{Alice}}, \pi^1_{\text{Bob}}) \in \Pi^g$ as well as $(\pi^2_{\text{Alice}}, \pi^1_{\text{Bob}}) \in \Pi^g$. We say that both programs $\pi^1_{\text{Alice}}, \pi^2_{\text{Alice}}$ are *goal-convergent*. The specification of goals is left to the specific formalism adopted by the chosen tool.

The introduction of the set Π^G is useful to clarify the notion of *misbehaviour*. A principal may run a program that deviates from the original protocol prescription, but if such deviation is irrelevant for the purpose of achieving the goal, the principal should not be considered as a misbehaving entity. This notion of misbehaviour contrasts from the usual interpretation that a principal misbehaves if she runs any program that differs from the expected one. However, our interpretation is necessary for accountability as in a dispute a judge should never blame a principal who runs a goal-convergent program.

Having seen the definition of a protocol, we can specify the definition of *verifiability test* as follows.

Definition 2 (Verifiability Test). *A* verifiability test $\mathtt{vt}(\mathcal{T}, \mathtt{g}) : \mathtt{bool}$ *is an efficient and terminating algorithm such that:*

- $\mathcal{T}$ *is a set of witness traces;*
- $\mathtt{g}$ *is a goal in* G.

The verifiability test should return `true` if, according to the evidence, a protocol run met the goal. It should return `false` otherwise. In other words, the verifiability test returns `true` if it accepts the set of witness traces, and `false` otherwise. Definition 3 formalises the concept of verifiability.

Definition 3 (g-verifiability). *A Protocol P is* g-verifiable *if P admits a verifiability test* vt *that meets the following conditions:*

1. *(soundness)* $\mathtt{vt}(\mathcal{T}, \mathtt{g}) : \mathtt{true} \implies \mathtt{g}$ *holds in $r(P)$;*
2. *(completeness)* $\mathtt{g}$ *holds in* $r(P) \implies \mathtt{vt}(\mathcal{T}, \mathtt{g}) : \mathtt{true}$*;*

for any run $r(P)$.

The soundness condition guarantees that the verifiability test returns `true` only if the goal holds in a run. However, this condition alone is not sufficient: a verifiability test that always returns `false` is sound but useless. Such kind of possibilities is ruled out with the completeness condition. Completeness implicitly states that the verifiability test cannot fail if all principals execute programs that converge towards the goal. It follows that P is correct as it meets the goal when all principals behave honestly.

Both soundness and completeness conditions can be checked automatically with cryptographic tools as reachability properties. For soundness, we check that there exists no trace in which we reach a state where the verifiability test returns `true` while the goal does not hold. For completeness, we check that there exists no trace in which we reach a state where the verifiability test returns `false` assuming all principals being honest. The analysis of three case studies considered later in this paper demonstrates that such mechanisation is possible.

Next, we focus on accountability, more precisely on a test that can be used to identify those principals who are responsible in the case a goal is not reached.

Definition 4 (Accountability Test). *An* accountability test $\mathtt{at}_{\alpha_\mathsf{y}}(\mathcal{T}, \mathtt{g}, \mathsf{A}) : \mathtt{bool}$ *is an efficient and terminating algorithm such that:*

- *$\mathcal{T}$ is a set of witness traces;*
- *$\mathtt{g}$ is a goal in G;*
- *α_y is an indicted principal over the set of principals A.*

The definition of the accountability test is methodologically close to the definition of verifiability. The test should return `true` if according to the witness traces the indicted principal α_y did not run a goal-convergent program, namely $\pi_{\alpha_i} \notin \Pi^{\mathtt{g}}_{\alpha_\mathsf{y}}$. The test should return `false` otherwise.

Now, we can advance a definition of accountability that is centred around a principal and a protocol's goal.

Definition 5 (($\alpha_\mathsf{y}, \mathtt{g}$)-accountability). *A Protocol P is* $(\alpha_\mathsf{y}, \mathtt{g})$-accountable *if given an indicted principal $\alpha_\mathsf{y} \in \mathsf{A}$, a goal $\mathtt{g}$, and the set of its goal-convergent programs $\Pi^{g}_{\alpha_\mathsf{y}}$, P meets the following conditions*

1. *P is $\mathtt{g}$-verifiable;*

and P admits an accountability test $\mathtt{at}_{\alpha_\mathsf{y}}$ that meets the following conditions:

2. *(soundness)* $\mathtt{at}_{\alpha_\mathsf{y}}(\mathcal{T}, \mathtt{g}) : \mathtt{false} \implies \pi_{\alpha_\mathsf{y}} \in \Pi^{\mathtt{g}}_{\alpha_\mathsf{y}}$*;*
3. *(completeness)* $\pi_{\alpha_\mathsf{y}} \in \Pi^{\mathtt{g}}_{\alpha_\mathsf{y}} \implies \mathtt{at}_{\alpha_\mathsf{y}}(\mathcal{T}, \mathtt{g}) : \mathtt{false}$.

for any run r(P).

Condition 1 guarantees that the event that triggers the failure can be identified, namely anyone can be convinced that a run of P failed to ensure the goal. The relation between verifiability and accountability becomes clear here. If a goal is not verifiable, then we cannot account any principal because we cannot state whether the protocol run met the goal or not. Hence, verifiability is a precondition for accountability. Note that condition 1 is goal-centred and independent from the indicted principal α_{y}.

Conditions 2 and 3 are defined similarly to the corresponding conditions for verifiability: the accountability test returns `true` if it accepts the set of witness traces, and `false` otherwise. Soundness guarantees that the accountability test returns `false` only if the indicted principal runs a goal-convergent program. Completeness states that the accountability test cannot return `true` if the indicted principal runs a goal-convergent program.

Remark. Verifiability is essential to have a meaningfulness definition of accountability. For example, let us assume a protocol P with three principals α_1, α_2, and α_3 of which only α_1 is partially g-accountable (i.e. P is partially (α_1, g)-accountable) according to conditions 2 and 3 only. If α_1 is not guilty (i.e. the accountability test fails), then we cannot say anything else about accountability in P without condition 1 since α_1 is the only culpable principal. In particular, we cannot say whether P failed because either or both α_1 and α_2 misbehaved or due to an external attacker. We cannot even say if the protocol meets the goal. If we can rule out the possibility for an external attacker, thanks to Condition 1, we know that at least either or both α_1 and α_2 misbehaved although P is neither (α_1, g)-accountable nor (α_2, g)-accountable, something that we would miss without Condition 1.

Finally, we propose the definition of *full* g*-accountability.* It states that a protocol is fully accountable for a goal if the protocol is accountable for each principal on that goal.

Definition 6 (Full g-accountability). *A Protocol P is* fully g-accountable *if $\forall \alpha \in \mathsf{A}$, P is (α, g)-accountable for any run r(P).*

It is easy to see that all three conditions in our definition of accountability can be checked automatically. Condition 1 regards verifiability, and we have already seen that soundness and completeness conditions of g-verifiability can be modelled as reachability properties to be automatically checked by cryptographic tools. Conditions 2 and 3 can also be modelled as reachability properties. In particular, soundness can be checked by showing that there exists no trace in which we reach a state where the accountability test returns `false` when all principals but the indicted are honest, and the verifiability test fails. For completeness, we check that there exists no trace in which we reach a state where the accountability test returns `true` when all principals but the indicted are dishonest.

Table 1 describes the systematic approach that can be used to check verifiability and accountability as reachability properties. This approach is validated

Table 1. Strategies to model verifiability and accountability as reachability properties.

Property	Condition	Principals controlled by the attacker	Strategy
Verifiability	Soundness	All (modulo the goal)	$\mathtt{vt}(\mathcal{T},\mathtt{g}):\mathtt{true} \rightsquigarrow$ `g_holds`
	Completeness	None	$\mathtt{vt}(\mathcal{T},\mathtt{g}):\mathtt{true}$
Accountability	Soundness	Indicted	$\mathtt{vt}(\mathcal{T},\mathtt{g}):\mathtt{false} \rightsquigarrow \mathtt{at}_{\alpha_y}(\mathcal{T},\mathtt{g}):\mathtt{true}$
	Completeness	All but the indicted	$\mathtt{at}_{\alpha_y}(\mathcal{T},\mathtt{g}):\mathtt{false}$

with two different tools and three different security protocols in the following sections.

4 Case Study I: Secure Exam Protocol

Bella et al. [18] propose a secure exam protocol that does not rely on any trusted party. Hence it can resist to corrupted candidates and authorities. The protocol involves four roles (i.e. candidate, administrator, examiner, and invigilator), and runs in four phases (i.e. preparation, testing, marking, and notification). The most interesting aspect of the protocol regards the outcome of preparation, in which candidate and administrator jointly generate the candidate's pseudonym as a pair of visual cryptography shares using an oblivious transfer scheme. One visual crypto share is held by the candidate, who prints it on a paper sheet together with signatures generated by the administrator. The other visual crypto share is printed by the administrator as a transparency printout and handed to the candidate at testing. Each share alone does not reveal the pseudonym, which the candidate learns only when the two shares are overlapped at testing. Thus, the goal of preparation is to distribute the generation of the two visual cryptography shares that, when overlapped, reveal an intelligible code. We consider this goal to analyse accountability, hence we leave testing, marking, and notification phases and focus only on the outcome of preparation in our analysis.

The idea underlying the preparation phase is that the candidate provides a commitment to an index into an array while the administrator fills the array with a secret permutation of the characters, and only when the two secrets are brought together is the selection of a character determined. Notably, no one learns anything about the code without the knowledge of both shares. The outcome of preparation is two sheets jointly generated by candidate and administrator using a combination of visual cryptography, commitment, and oblivious transfer schemes. The commitment scheme comes with a function $commit(\cdot,\cdot)$ that takes in a random value and a secret, and outputs the commitment. The oblivious transfer schemes consists of (i) the function $obf(\cdot,\cdot)$, which takes a commitment and a set of secrets, and returns a set of obfuscated values; and (ii) the function $deobf(\cdot, \cdot)$, which takes in a set of obfuscated values and a commitment, and returns the corresponding set of secrets.

The authors prove in ProVerif that the protocol meets a set of authentication, privacy, and verifiability properties. They also prove a form of accountability, but their definitions are specific to the protocol in question. Differently, we prove accountability using our general approach that can be applied to other protocols.

As we shall see later, our accountability tests take in the content of the paper and transparency sheets. The paper sheet contains the candidate visual share β, the set of candidate's chosen indexes I, the random commitment value c, and two signatures $sign1$ and $sign2$ both generated by the administrator and encoded as QR codes. The first signature contains the commitment com_A of the administrator on α. The second signature contains the commitment com_C of the candidate chosen indexes I, and the set of obfuscated values Ω due to the oblivious transfer scheme. The transparency printout contains the visual share α, and the random commitment value a on the administrator's commitment com_A. A succinct representation of the contents of the sheets is outlined in Table 2.

Table 2. The content of the paper and transparency sheets

Sheet	Content		Description
Paper (candidate)	β		Visual cryptography share
	c		Random commitment value on com_c
	I		Set of indexes chosen by the candidate
	$sign1$	com_A	Administrator's commitment
	$sign2$	com_C	Candidate's commitment on I
		Ω	Set of obfuscated values
Transparency (administrator)	α		Visual cryptography share
	a		Random commitment value on com_a

4.1 Analysis

Our analysis focuses on the goal of generating two correct visual shares. If so, an intelligible code should appear when the candidate overlaps paper and transparency sheets. We propose two distinct dispute resolution procedures (i.e. accountability tests) — one for the candidate and the other for the administrator — for which we can have formal guarantees of correctness. Our accountability tests can be used with the same sheets generated at preparation of the original protocol. We use ProVerif, an automatic protocol analyser that can prove reachability and equivalence-based properties in the Dolev-Yao model.

Verifiability. First, we demonstrate that the protocol is g-verifiable. Namely, there exists a verifiability test that is sound and complete according to Definition 3 for the goal of generating an intelligible pseudonym. We specify verifiability in ProVerif as a reachability property and use correspondence assertions to prove soundness. The verification scenario consists of checking that, for any execution of the protocol, all traces in which the verifiability test returns `true`, there is another event, earlier in the trace, that signals that the goal holds. In this case, the goal holds if both candidate and administrator print the correct visual shares on the respective sheets. The attacker may control either the administrator or the candidate, but we force two events to be emitted by candidate and administrator processes only when they print the correct visual shares. ProVerif proves that there exists no trace in which the attacker can input the verifiability test with false data so that the test returns `true` without that the goal holds.

The verification scenario to prove completeness consists of checking that, for any execution of the protocol in which the goal holds, the verifiability test returns `true`. In this case, the ProVerif model enforces only honest principals and prevents the attacker to manipulate the input data of the verifiability-tests. In fact, a complete verifiability-test must succeed if its input data is correct. Specifically, the overlapping of the two visual shares should always produce an intelligible code. ProVerif proves that there exists no trace in which the verifiability test returns `false` when its input data is correct. Since the proposed verifiability test is sound and complete, the protocol is g-verifiable.

Algorithm 1. The accountability test for the Candidate

```
Data:
  - $paper = \beta, c, I, sign1, sign2$ where
    - $sign1 = Sign_A\{com_A\}$.
    - $sign2 = Sign_A\{com_C, \Omega\}$.
  - $transp = \alpha, a$.

if $sign1 = \perp$ or $sign2 = \perp$ or $com_c \neq commit(c, I)$ or $\beta \neq deobf(\Omega, c)$ then
    return true
else
    return false
```

Accountability. We propose Algorithms 1 and 2 as accountability tests for candidate and administrator respectively. In the following, we show that both algorithms enable the protocol to meet soundness and completeness according Definition 5.

Accountability can be specified as reachability property, but the verification scenario to check the soundness of the accountability test differs from the one we have seen for verifiability. To check soundness, we leave the indicted principal under the control of the attacker (i.e., we force all principals but the indicted one to be honest). Then, if the protocol fails (i.e. the verifiability test returns

Algorithm 2. The accountability test for the Administrator

```
Data:
  - $paper = \beta, c, I, sign1, sign2$ where
    - $sign1 = Sign_A\{com_A\}$.
    - $sign2 = Sign_A\{com_C, \Omega\}$.
  - $transp = \alpha, a$.

if $sign1 \neq \bot$ and $sign2 \neq \bot$ and $com_A \neq commit(a, \alpha)$ then
    return true
else
    return false
```

false, we expect that the accountability test returns true, namely it blames the indicted principal for all traces and protocol runs. ProVerif proves that Algorithms 1 and 2 are sound since there exists no trace in which the accountability tests return false in such scenario.

The verification scenario to prove completeness is complementary to the scenario outlined above. We assume the indicted principal to be honest and leave all the others principals under the control of the attacker. We expect that the accountability test does not blame the indicted principal, hence it returns false for all traces and protocol run. ProVerif proves that Algorithms 1 and 2 are complete since there exists no trace in which the accountability test returns true in this verification scenario. Thus, we conclude that the secure exam protocol is verifiable and accountable for the goal of correctly generating and distributing the visual cryptography shares.

5 Case Study II: Certificate Transparency

Public Key Infrastructures (PKI) are the source of accountability for a very common use case: a client C – who wants to establish a secure connection to a server S – receives and checks a certificate issued by a certificate authority CA. A certificate essentially binds an identity S with a public key PK_S, along with other information such as the expiration date and a chain of certificates leading to a root CA, which we denote as $cert_S = sign_{CA}(PK_S, S, info)$. The strongest limitation of PKI is that the client should maintain a list of all trusted CAs (usually hundreds) and if even one of them becomes compromised and misbehaves, then the whole system is compromised. In fact, a dishonest server colluding with a compromised CA can obtain a signed certificate for another server identity and impersonate them, and this behaviour can go undetected since the dishonest server can show the certificate only to the targeted users. Moreover, the PKI standard does not require a CA to show which certificates it has issued.

5.1 Certificate Transparency

To solve the accountability problem of PKIs, Google proposed Certificate Transparency (CT) [19], an extension of standard PKIs that allows the servers to check that the CAs behave properly. It does so by maintaining a *public, append-only, cryptographic log of issued certificates* that anybody can check. When a client C wants to connect to S, she first receives from S their certificate $cert_S$, along with a cryptographic proof that $cert_S$ is included the log L. Conversely, the log can be audited either as a whole in a heavy-weight fashion, or in small parts by piggy backing a chatter protocol on top of the handshake between C and S, as shown with by the “cloud” in the communication diagram of Fig. 1.

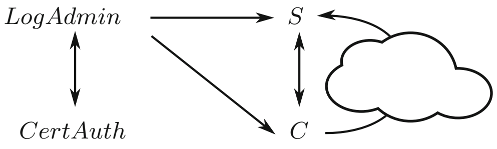

Fig. 1. Certificate Transparency, communication diagram

If a CA misbehaves then their misconduct will appear publicly in the log and will be revealed by auditing. On the other side, a log administrator colluding with a CA could produce two different histories for the client and the server: it could, for example, give S a log where there appears no fake certificate for S, and give C a log where such fake certificate appears, in order to convince her to connect to a rouge server. However, the log signs all histories presented to the various stakeholders, so if the log gives incompatible histories to different entities, their misbehaviour will eventually be detected, by comparing two incompatible histories. All these operations can be implemented efficiently by the use of Merkle trees [20], i.e. they require time and space $O(log(n))$ for n certificates in the log.

It is important to note that CT does not prevent attacks against clients: a CT-enabled client C checks validity and the presence of a certificate in a log, but nothing prevents C from accessing a compromised server S_D if both the validity check and the presence check succeed. CT ensures instead that—as long as the Log Administrators are honest—*eventually* the presence of a fake certificate is revealed to the legitimate owner of a certain domain, who can then take appropriate actions to contain the breach. Furthermore, it claims to support accountability for both the CA and the log administrator, in that if an attack happens there is evidence that they misbehaved: for the CA, this is the presence of a signed certificate without a proper proof of identity given by the legitimate owner, while for the Log Administrator, it is the presence of two incompatible histories, eventually revealed by two different parties exchanging them.

5.2 Analysis

We construct a symbolic model of the Certificate Transparency protocol and show that it satisfies both verifiability and accountability for the Certificate Authorities and the Log Administrators. We model the protocol with AIF-ω [4], which allows to encode stateful protocols by tracking the membership of values in a number of sets, indexed by agent names and other parameters. For example, the logs in our model are represented by a family of sets $log(LogAdmin, Server, Cert, Auth, User)$ of public keys, logged by a $LogAdmin$, issued for a $Server$ by a $CertAuth$, and presented to an $User$, where each camel-case word defines the respective role in the protocol. Therefore we allow an $la \in LogAdmin$ to present two different stories to two different users and treat a log as a database of signed public keys, related to the CAs that produced them and the servers that they represent. We abstract away from the implementation details using Merkle trees and enforce that their properties–efficient querying for the presence of a certificate, and efficient proofs of extension–are maintained in the database.

Verifiability. We show first that the protocol is g-verifiable according to Definition 3 for the goal of producing a valid certificate for a server. The soundness result specifies that, for all execution traces where the verifiability test succeeds, then the protocol has been executed only by an honest and behaving certificate authority and log administrator; in other words, there has not been a trace where a misbehaving CA or LA manage to pass the verifiability test. In this case, the goal holds if there is no scenario where a malicious certificate is produced that does not come with a proof of identity for its server and does not show two incompatible logs (i.e. one with the certificate and one missing it). Algorithms 3 and 4 check these conditions. That is, Algorithm 3 returns `true` if and only if the Certificate Authority lacks a valid proof of identity for the given combination of server and public key, and Algorithm 4 returns `true` iff there is evidence that the Log Administrator produces two logs log_1 and log_2 that are incompatible extensions of one another. It is important to stress that in this model we assume that there is a direct connection between the interested Server and Client comparing the two logs, whereas in reality there is an indirect channel realised by the chatter network, as shown in Fig. 1. Hence in the model, the check of Algorithm 4 is quantified over all possible pairs of logs, and this is not a problem for soundness, but for completeness it requires further justification.

Completeness requires that if a certificate is produced by a dishonest CA and logged only for the client by a dishonest LA, then the client and the server, communicating through the chatter network, will be able to discover the misbehaviour. In this case it is important to stress that this is a reachability property, in that *eventually* the client and server will be able to discover the misbehaviour through the chatter log, but that might be after the client has suffered a man-in-the-middle attack.

Accountability. The accountability test for the Certificate Authority and the Log Administrator coincide with Algorithms 3 and 4 considered singularly. From our

Algorithm 3. The accountability test for the Certificate Authority

```
Data:
  - $cert = sign_{CA}(PK, S, info)$
  - $poi = proofOfID(PK', S')$

if $poi = \perp$ or $PK \neq PK'$ or $S \neq S'$ then
    return true
else
    return false
```

Algorithm 4. The accountability test for the Log Administrator

```
Data:
  - $log_1$
  - $log_2$.

if $log_1 \not\preceq log_2$ and $log_2 \not\preceq log_1$ then
    return false
else
    return true
```

model, we prove that both algorithms are sound and complete: they do no blame any honest CA/LA, while in any case of a misbehaving CA/LA, there is a proof that they misbehaved.

It is interesting to note that the accountability test amounts to splitting the two checks of the verifiability test, which are aimed at indicting the Certificate Authority and the Log Administrator, respectively. In fact, if a protocol is fully accountable, i.e. if for every principal the accountability test is both sound and complete, we can produce a verifiability test by composing the accountability tests of each principal, therefore obtaining a verifiability test that is also both sound and complete.

6 Case Study III: Bingo Voting

Bingo Voting is a cryptographic voting scheme proposed by Bohli, Müller-Quade and Röhrich in 2007 [21]. It provides individual verifiability based on a trusted random number generator. Each voter receives a receipt that enables the voter to verify that the corresponding vote was counted correctly. But the receipt does not provide any information about how the voter voted to any third party. The original version of Bingo Voting does not include any dispute resolution procedure that enables voters to prove that some manipulation took place and their vote was altered. Küsters et al. [2] demonstrated that the original version also allows dishonest voters to spoil an election by wrongly complaining that the election was manipulated even if this is not the case. Bohli et al. [22] then proposed some improvements to the original scheme that enable dispute resolution procedures during the voting and the tallying phases. In this paper, we consider the improved version of Bingo Voting and focus on the dispute resolution procedure during the voting phase concerning the *cast-as-intended* goal.

The underlying idea of Bingo Voting is that the voting machine encodes the voter's choice in the receipt using random numbers. Each receipt in this election contains each candidate and one random number assigned to it. There are two types of random numbers, *dummy random numbers* generated by the voting authority before the voting phase, and *fresh random numbers* that are generated during the voting phase by the trusted random number generator. The random number used to denote the voter's choice is the fresh random number generated and displayed by the trusted random number generator inside the voting booth. All other random numbers associated to the rest of candidates are dummy random numbers.

At preparation phase, the voting authority generates and publish the set of dummy votes. A dummy vote consists of a pair of Pedersen commitments that hide a dummy random number and the corresponding candidate. The voting authority generates a number of dummy votes equal to the product of the number of candidates and the number of the voters. In addition to the set of dummy votes, the voting machine generates a proof using randomized partial checking [23] to show that each candidate has received the same number of dummy votes. At voting, the voter enters the voting booth and records her choice on a paper ballot that is then fed into the scanner of the voting machine. The scanner prints a random barcode onto one margin of the paper ballot. The barcode is used as alignment information in case of a dispute as the receipt contains an identical barcode. The trusted random number generator generates one fresh random number that is sent to the voting machine and displayed on a screen inside the voting booth so that the voter sees the number. The voting machine generates a receipt such that the fresh random number generated by the trusted random generator is printed next to the candidate chosen by the voter, while unused dummy random numbers are printed next to the other candidates. If the voter thinks that the receipt is correct, she destroys the paper ballot and leaves the voting booth keeping the receipt. The paper ballot needs to be destroyed to prevent vote-buying and coercion.

In the case of a dispute, the voter can put paper ballot and receipt inside privacy sleeves. The privacy sleeves aim at solving a dispute without revealing how a voter voted. There are two types of privacy sleeves. The first type leaves uncovered the barcodes and the candidate names (see Fig. 2). This would allow the voter to prove that the candidates are not placed identically with respect to the barcode on the receipt and the paper ballot. The second type of sleeve uncovers the barcodes and one row of the marking area on the paper ballot and of random numbers on the receipt (see Fig. 3). This would allow the voter to prove a mismatch between her choice and the random numbers that appear on the receipt and on the screen.

The next phase of Bingo Voting is the tallying phase, which we do not cover here since our focus is on the dispute resolution procedure during the voting phase.

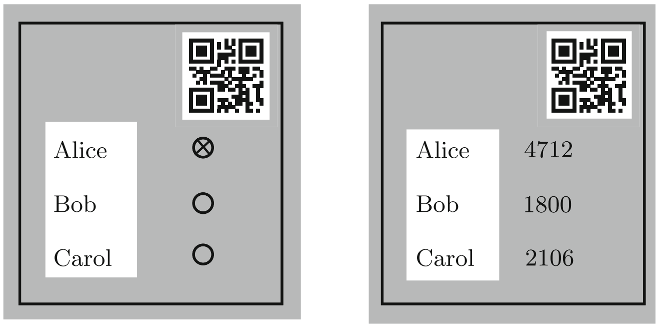

Fig. 2. The type of privacy sleeve to check the correctness of the alignment of candidates

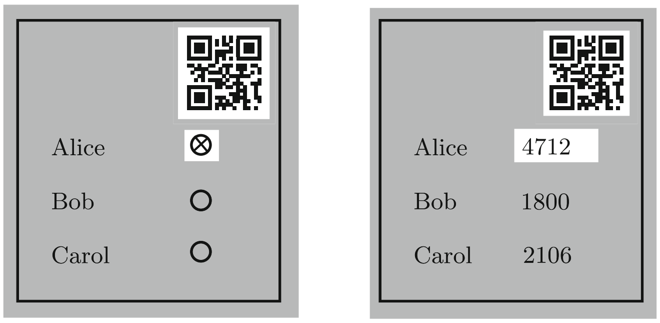

Fig. 3. The type of privacy sleeve to check the correctness of the encoding of the voter's choice

6.1 Analysis

We analyse verifiability and accountability of the improved version of Bingo Voting in ProVerif. We assume two candidates are competing for the election and, as we shall see later, prove that the improved version of Bingo Voting meets verifiability but not accountability.

Verifiability. Our analysis strategy to check verifiability of Bingo Voting is similar to the one adopted to check verifiability of the secure exam protocol in Sect. 4. We prove soundness using correspondence assertions and checking that, for any execution of the protocol, all traces in which the test returns `true`, the vote was cast as intended. Since either the voting authority/machine or the candidate can be malicious, we prove the soundness of the verifiability test in each of these scenarios. The verifiability test is as in Algorithm 5. It takes in the *paper* ballot, the *receipt* and the random number displayed in the *screen*, and checks whether the barcodes of paper ballot and receipt match. It also checks that the choice on the paper ballot is associated to the correct random number, which matches with the one displayed in the screen. ProVerif proves that the verifiability test is sound. To prove completeness, we check in ProVerif that when the input data of the test is correct, the test never returns `false`. Since the verifiability test is sound and complete, Bingo Voting allows any one to verify that a vote has been cast as intended.

Accountability. The accountability test for the dispute resolution coincides with the verifiability test. In fact, the failure of the verifiability test (i.e., it returns `false`) is a sufficient condition to blame the Voting Authority since the random generator is trusted by assumption. Hence, the soundness of the accountability test can be trivially checked by proving that if the verifiability test fails, then the accountability test never returns `false` when the indicted principal is the Voting Authority and it is controlled by the attacker. Since the accountability test and the verifiability test coincide, ProVerif trivially proves the soundness of the accountability test. To check completeness, we set the Voting Authority honest and the voter controlled by the attacker. We aim at showing in ProVerif that the accountability test never returns `true`, namely it does not blame the honest Voting Authority. ProVerif fails to prove completeness and shows an attack trace in which two corrupted voters can collaborate to falsely blame an honest Voting Authority. The attack consists of a voter who hands his receipt to the next voter. The latter, on the isolation assumption of the voting booth, swaps the fresh receipt printed by the Voting Machine with the one handed previously by the colluding voter. Then, he puts the fresh paper ballot and the old receipt in the privacy sleeves so that the two barcodes mismatch. The attack is meaningful unless voters are searched before entering the voting booth. We believe this is unlikely to happen as it would decrease the applicability and acceptance of the voting system. Thus, the improved version of Bingo Voting still fails in that of allowing dishonest voters to spoil an election by wrongly complaining that the election was manipulated even if this is not the case.

Algorithm 5. The verifiability test for Bingo Voting

```
Data:
  - $screen = r$.
  - $paper = choice, barcode\_p$.
  - $receipt = r1, r2, barcode\_r$.

if $(choice = c1$ and $r = r1$ and $barcode\_p = barcode\_r)$ or $(choice = c2$ and
 $r = r2$ and $barcode\_p = barcode\_r)$ then
    return true
else
    return false
```

7 Conclusion

Accountability is an essential property for critical systems. Although it has been studied in several security protocols, it has never been defined in a way that fully enables its automated analysis with cryptographic tools. To the best of our knowledge, we advance the first approach that enables the accountability analysis of security protocols automatically. Soundness and completeness conditions are tailored so that verifiability and accountability can be specified as reachability, a property that many cryptographic tools can check automatically nowadays. We validate our approach by applying our definitions to the analysis of three different protocols: a secure exam protocol, Certificate Transparency, and Bingo Voting. We propose the accountability tests that make exam administrators and candidates accountable for the failure of the exam. We show in ProVerif that our accountability tests are sound and complete. We prove in AIF-ω that Certificate Transparency meets its goal of blaming Certificate Authorities and Log administrators if they misbehave. Finally, we find that the improved version of Bingo Voting does not satisfy accountability, although we consider a trusted random generator.

Extending the applicability of automated verification methods to security protocols is a major direction for future work. Manual proofs are complicated and hard to follow as they may involve reasoning about probability and computational complexity, hence prone to human errors. We believe that our mechanised approach will favour the adoption of automated formal verification techniques for the analysis of accountability.

References

1. Cortier, V., Galindo, D., Küsters, R., Müller, J., Truderung, T.: SoK: verifiability notions for e-voting protocols. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 779–798 (2016)
2. Küsters, R., Truderung, T., Vogt, A.: Accountability: definition and relationship to verifiability. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, pp. 526–535. ACM, New York (2010)
3. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: CSFW, pp. 82–96. IEEE Computer Society (2001)
4. Mödersheim, S., Bruni, A.: AIF-ω: set-based protocol abstraction with countable families. In: Piessens, F., Viganò, L. (eds.) POST 2016. LNCS, vol. 9635, pp. 233–253. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49635-0_12
5. Milner, K., Cremers, C.J.F., Yu, J., Ryan, M.: Automatically detecting the misuse of secrets: Foundations, design principles, and applications. IACR Cryptol. ePrint Arch. 234 (2017)
6. Jagadeesan, R., Jeffrey, A., Pitcher, C., Riely, J.: Towards a theory of accountability and audit. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 152–167. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04444-1_10
7. Bella, G., Paulson, L.C.: Accountability protocols: formalized and verified. ACM Trans. Inf. Syst. Secur. **9**, 138–161 (2006)
8. Zhou, J., Gollmann, D.: A fair non-repudiation protocol. In: Proceedings of the 1996 IEEE Conference on Security and Privacy, SP 1996, pp. 55–61. IEEE Computer Society, Washington, DC (1996)
9. Abadi, M., Blanchet, B.: Computer-assisted verification of a protocol for certified email. In: Cousot, R. (ed.) SAS 2003. LNCS, vol. 2694, pp. 316–335. Springer, Heidelberg (2003). doi:10.1007/3-540-44898-5_17

10. Kremer, S., Ryan, M., Smyth, B.: Election verifiability in electronic voting protocols. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 389–404. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15497-3_24
11. Benaloh, J., Tuinstra, D.: Receipt-free secret-ballot elections (extended abstract). In: Proceedings of the 26th Symposium on Theory of Computing (STOC 1994), pp. 544–553. ACM, New York (1994)
12. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 539–556. Springer, Heidelberg (2000). doi:10.1007/3-540-45539-6_38
13. Cohen, J., Fischer, M.: A robust and verifiable cryptographically secure election scheme (extended abstract). In: Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS 1985), Portland, Oregon, USA, pp. 372–382. IEEE Computer Society (1985)
14. Benaloh, J.: Verifiable Secret-Ballot Elections. Ph.D. thesis, Yale University (1996)
15. Smyth, B., Ryan, M., Kremer, S., Kourjieh, M.: Towards automatic analysis of election verifiability properties. In: Armando, A., Lowe, G. (eds.) ARSPA-WITS 2010. LNCS, vol. 6186, pp. 146–163. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16074-5_11
16. Dreier, J., Jonker, H., Lafourcade, P.: Defining verifiability in e-auction protocols. In: Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2013), Hangzhou, China, pp. 547–552. ACM (2013)
17. Guts, N., Fournet, C., Zappa Nardelli, F.: Reliable evidence: auditability by typing. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 168–183. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04444-1_11
18. Bella, G., Giustolisi, R., Lenzini, G., Ryan, P.Y.: Trustworthy exams without trusted parties. Comput. Secur. **67**, 291–307 (2017)
19. Laurie, B., Langley, A., Kasper, E.: Certificate transparency. Technical report (2013)
20. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). doi:10.1007/3-540-48184-2_32
21. Bohli, J.-M., Müller-Quade, J., Röhrich, S.: Bingo voting: secure and coercion-free voting using a trusted random number generator. In: Alkassar, A., Volkamer, M. (eds.) Vote-ID 2007. LNCS, vol. 4896, pp. 111–124. Springer, Heidelberg (2007). doi:10.1007/978-3-540-77493-8_10
22. Bohli, J.M., Henrich, C., Kempka, C., Muller-Quade, J., Rohrich, S.: Enhancing electronic voting machines on the example of bingo voting. IEEE Trans. Inf. Forensics Secur. **4**, 745–750 (2009)
23. Jakobsson, M., Juels, A., Rivest, R.L.: Making mix nets robust for electronic voting by randomized partial checking. In: Proceedings of the 11th USENIX Security Symposium, Berkeley, CA, USA, pp. 339–353. USENIX Association (2002)