Verifying Accountability for Unbounded Sets of Participants

Kevin Morio and Robert Künnemann
CISPA Helmholtz Center for Information Security
Saarland Informatics Campus, Germany

***Abstract*—Little can be achieved in the design of security protocols without trusting at least some participants. This trust should be justified or, at the very least, subject to examination. One way to strengthen trustworthiness is to hold parties accountable for their actions, as this provides a strong incentive to refrain from malicious behavior. This has led to an increased interest in accountability in the design of security protocols.**

In this work, we combine the accountability definition of Künnemann, Esiyok, and Backes [21] with the notion of case tests to extend its applicability to protocols with unbounded sets of participants. We propose a general construction of verdict functions and a set of verification conditions that achieve soundness and completeness.

Expressing the verification conditions in terms of trace properties allows us to extend TAMARIN—a protocol verification tool—with the ability to analyze and verify accountability properties in a highly automated way. In contrast to prior work, our approach is significantly more flexible and applicable to a wider range of protocols.

I. INTRODUCTION

Holding parties accountable for their misconduct—most often detection is deterrent enough—is an incentive to avoid malicious behavior from the outset. Participants have to weigh up whether an action is worth the consequences. Accountability is applicable to a wide range of protocols, such as e-voting, electronic payment processing, and electronic health care transactions.

Künnemann, Esiyok, and Backes [21] proposed a protocol-agnostic definition of accountability and an automated verification technique. They consider accountability a meta-property defined with respect to a security property φ. A protocol that provides accountability for φ provides the information necessary to determine whether φ has been violated and, if so, which parties should be held accountable. The *verdict* contains all groups of parties that are (jointly) accountable for a violation. Verdicts are returned by a total function—the *verdict function*—given the trace of a protocol execution.

To prove that a specified verdict function provides the protocol with accountability for a security property, a set of verification conditions must be verified. Künnemann, Esiyok, and Backes [21] show that their proposed verification conditions are sound and complete: If and only if all conditions hold, accountability is provided. Expressing the verification conditions as trace properties allows them to exploit existing protocol verification tools and achieve a high degree of automation.

However, the verdict function and verification conditions they propose require the parties to be explicitly stated in each verdict, thus inherently limiting the set of parties that can be blamed. This restricts the expressiveness of the approach as in almost all real-world protocols the same party can be involved in multiple parallel sessions (e.g., TLS) or the number of participants is not known a priori and can change dynamically during the protocol execution (e.g., the Signal chat protocol).

In this work, we address this shortcoming by combining their approach with the notion of case tests—an idea inspired by the accountability tests of Bruni, Giustolisi, and Schuermann [17]. Case tests are trace properties with free variables, where each free variable stands for a party that should be blamed for a violation. This is in contrast to accountability tests, where one test applies to one party and joint accountability is not expressible. In contrast to an explicitly stated verdict function, case tests can match multiple parts of a trace. The verdict function is thus implicitly defined as the union of groups of parties blamed by instances of the case tests. This approach provides flexibility and allows—for the first time—the analysis of protocols with unbounded sets of participants. It also improves readability, as intuitively, each case test stands for a specific way a violation can be triggered, in contrast to the previous, explicit formulation of the verdict function, in which all combinations of actions that constitute a violation had to be captured.

We list our contributions as follows.

1) We derive a set of verdict-based verification conditions based on the accountability definition of Künnemann, Esiyok, and Backes [21] and show that they provide soundness and completeness.
2) We introduce the notion of case tests and use them to define verdict functions that are significantly more flexible than previous ones.
3) We show how the verification conditions can be rewritten using case tests and formalize requirements to encode them in terms of trace properties. By proving their relation to the verdict-based variant, we can transfer their soundness and completeness to the encoded verification conditions.
4) We implemented our approach in TAMARIN by adding the ability to define accountability lemmas and case tests.
5) We showcase our methodology by extending previous models of OCSP Stapling [26], [27], and Certificate Transparency [28] to an unbounded number of parties, and applying it to new models of mixnets and the e-voting protocol MixVote/Alethea [19], [22].

The paper is structured as follows. Related work is discussed in Section II and the accountability definition of Künnemann, Esiyok, and Backes [21] is elucidated in Section III. We show a sound and complete decomposition of accountability into verification conditions in Section IV and introduce case tests to define a general verdict function in Section V. We elaborate on the counterfactual relation in Section VI and present the verification conditions expressed as trace properties in Section VII. In Section VIII, we explain the implementation in TAMARIN. In Section IX, we describe the case studies, evaluate our verification results, and compare them with the results in the framework of [21]. We conclude in Section X.

II. Related Work

In the security setting considered in this work, we regard accountability as the ability to identify malicious parties. Different approaches and notions of accountability have been proposed. However, in previous works, these are only described informally or tailored to specific protocols and security properties [3], [4], [11], [15]. An emerging problem is the difficulty of defining when a party's behavior should be considered *malicious* and the implications this has on completeness—holding *all* malicious parties accountable.[1]

In the past, misbehaving and dishonest parties were treated as equivalent. While this is a reasonable approximation for some cryptographic tasks—for example, secure multi-party computation—it is not suitable in the context of accountability. Completeness would require identifying all dishonest parties, but a dishonest party does not have to deviate or may behave in a way that is indistinguishable from the protocol. Some approaches [5], [6] in the distributed setting assume that all communication is observable and classify any trace not producible by honest parties as malicious behavior. In the security setting, this assumption is impossible to satisfy and the definition of malicious behavior unreasonable, as parties may communicate through hidden channels and deviate in harmless ways.

Haeberlen, Kouznetsov, and Druschel [5] propose PeerReview, a system which can detect Byzantine faults in the distributed setting. The system requires that all communication is observable which is a suitable assumption in a distributed environment but unrealistic in the security setting.

Jagadeesan, Jeffrey, Pitcher, *et al.* [6] provide multiple general notions of accountability based on an abstract labeled transition system in the distributed setting. However, the authors admit that "the only auditor capable of providing [completeness] is one which blames all principals who are capable of dishonesty, regardless of whether they acted dishonestly or not."

Küsters, Truderung, and Vogt [8] define accountability in the symbolic and computational model using accountability properties. These are specified in a formal language. Künnemann, Garg, and Backes [20] point out that these properties are not expressive enough. Furthermore, they identify significant weaknesses in the case of joint misbehavior.

Another approach is to consider protocol actions as the actual causes for security violations [9], [13], [14]. However, protocol actions may be causally related to a security violation but still be harmless and without any malicious intent.

In recent work, Künnemann, Garg, and Backes [20] propose a general protocol agnostic definition of accountability in which the fact that a party deviated is considered a potential cause for a security violation. Based on this approach, Künnemann, Esiyok, and Backes [21] provide an automated verification technique in the single-adversary setting. They define the *a posteriori verdict* (apv), which given a trace of a protocol execution, returns all groups of parties that are jointly accountable for the security violation. If there exists a function—called the verdict function—which coincides with the apv for all traces of the protocol, the verdict function is said to provide the protocol with accountability for a specified security property. In their work, the verdict function uses a case distinction over traces and specifies a verdict per case. This form requires that the parties be explicitly stated in a verdict and thus fixes the number of parties. Most protocols have a fixed number of *roles*, but the same party can run many sessions with different communication partners (e.g., several servers in TLS, or partners in chat protocols).

Bruni, Giustolisi, and Schuermann [17] give a definition of accountability based on the existence of per-party accountability tests which decide whether the party should be held accountable for a violation. A verdict is obtained by considering all parties for which their test is positive as singleton sets. Since each singleton verdict contains exactly one party, joint accountability is not expressible. Moreover, as noted by Künnemann, Esiyok, and Backes [21], there are some flaws in the criteria of their definition allowing parties to be blamed even if the security of a protocol cannot be violated or violations remain undetected under certain circumstances.

III. Background

We provide an overview of the notation and concepts we use throughout this work and recall the accountability definition of Künnemann, Esiyok, and Backes [21].

A. Preliminaries

a) Sets, sequences, and multisets: We denote the set of integers $\{1, \ldots, n\}$ by $[n]$, the power set of S by 2^S and the set of finite sequences of elements from S by S^*. For a sequence s, we write s_i for the i-th element, $|s|$ for the length of s, and $idx(x) \coloneqq \{1, \ldots, |s|\}$ for the set of indices of s. We write $\vec{s}$ to emphasize that s is a sequence.

b) Terms: Cryptographic messages are modeled as abstract terms. We specify an order-sorted term algebra with the sort msg and two incomparable subsorts pub and $fresh$ for two countably infinite sets of public names (PN) and fresh names (FN). We assume pairwise disjoint, countably infinite sets of variables $\mathcal{V}_s$ for each sort s. The set of all variables $\mathcal{V}$ is the union of the set of variables for all sorts $\mathcal{V}_s$. We write $u\colon s$

[1] Note that this notion of completeness concerns what we would consider the verdict. This is opposed to the completeness of the verification conditions with respect to the accountability of a protocol, saying that all protocols that provide accountability are recognized as correct by the verification conditions.

when the name or variable u is of sort s. A signature Σ is a set of function symbols, each with an arity. We write f/n for a function symbol f with arity n. The set of well-sorted terms constructed over Σ, PN, FN, and $\mathcal{V}$ is denoted by $\mathcal{T}_\Sigma$. The subset of ground terms—terms without variables—is denoted by $\mathcal{M}_\Sigma$. If Σ can be inferred from context, we write $\mathcal{T}$ and $\mathcal{M}$ respectively.

c) Equational theories: An *equation* over the signature Σ is an unordered pair $\{s, t\}$ of terms t, $s \in \mathcal{T}_\Sigma$, written $s \backsimeq t$ or $s = t$ when the meaning can be inferred from context. Equality is defined with respect to an equational theory E, a binary relation $=_E$ induced by a finite set of equations which is closed under the application of function symbols, bijective renaming of names, and substitution of variables by terms of the same sort. An equational theory E formalizes the semantics of the function symbols in Σ. We say that two terms t and s are *equal modulo* E iff $t =_E s$. Set membership modulo E is denoted by $\in_E$ and defined as $e \in_E S$ iff $\exists e' \in S.\ e' =_E e$. The usual operations on sets modulo E are defined accordingly.

d) Facts: We assume an unsorted signature Σ_{fact} which is disjoint from Σ. The set of *facts* is defined by

$$\mathcal{F} \coloneqq \left\{ \mathsf{F}(t_1, \ldots, t_n) \;\middle|\; t_i \in \mathcal{T}, \mathsf{F} \in \Sigma^k_{\text{fact}} \right\},$$

where Σ^k_{fact} denotes all function symbols of arity k in Σ_{fact}. The set of *ground facts* is denoted by $\mathcal{G}$.

e) Substitutions: A *substitution* σ is a well-sorted function from variables $\mathcal{V}$ to terms $\mathcal{T}_\Sigma$ that corresponds to the identity function on all variables except on a finite set of variables. Overloading notation, we call this finite set of variables the *domain* of σ, which we denote by $\mathrm{dom}(\sigma)$. The image of $\mathrm{dom}(\sigma)$ under σ is denoted by $\mathrm{rng}(\sigma)$. For the homomorphic extension of σ to a term t or a trace formula φ, we write $t\sigma$ and $\varphi\sigma$ respectively. We write $\sigma\big[v \mapsto w\big]$ to denote the update of σ at v such that $\sigma\big[v \mapsto w\big](x) = w$ for $x = v$ and $\sigma(x)$ otherwise. If σ is injective, we denote its inverse by σ^{-1}. We say that two substitutions σ, σ' are equal modulo E if $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma')$ and $\sigma(x) = \sigma(x')$ for all x in $\mathrm{dom}(\sigma)$.

f) Valuation: Each sort s is associated with a domain $\mathbf{D}_s$. The domain for temporal variables is the rational numbers $\mathbf{D}_{temp} \coloneqq \mathbb{Q}$ and the domains for messages are $\mathbf{D}_{msg} \coloneqq \mathcal{M}$, $\mathbf{D}_{fresh} \coloneqq FN$, and $\mathbf{D}_{pub} \coloneqq PN$. A function θ from $\mathcal{V}$ to $\mathbb{Q} \cup \mathcal{M}$ is a *valuation* if it respects sorts, that is, $\theta(\mathcal{V}_s) \subseteq \mathbf{D}_s$ for all sorts s. We write $t\theta$ for the homomorphic extension of θ to a term t.

g) Trace properties: Trace properties are sets of traces which are specified by trace formulas in a two-sorted first-order logic which supports quantification over messages and timepoints.

Definition 1 (Trace formula)**.** *A trace atom is either false $\bot$, a term equality $t_1 \approx t_2$, a timepoint ordering $i \lessdot j$, a timepoint equality $i \doteq j$, or an action $\mathsf{F}@i$ for a fact $\mathsf{F} \in \mathcal{F}$ and a timepoint i. A trace formula is a first-order formula over trace atoms.*

Definition 2 (Satisfaction relation)**.** *The satisfaction relation $(tr, \theta) \vDash \varphi$ between a trace tr, a valuation θ, and a trace formula φ is defined as follows.*

$$\begin{aligned}
&(tr,\theta) \vDash \bot && \textit{never}\\
&(tr,\theta) \vDash \mathsf{F}@i && \iff \theta(i) \in idx(tr) \wedge \mathsf{F}\theta \in_E tr_{\theta(i)}\\
&(tr,\theta) \vDash i \lessdot j && \iff \theta(i) < \theta(j)\\
&(tr,\theta) \vDash i \doteq j && \iff \theta(i) = \theta(j)\\
&(tr,\theta) \vDash t_1 \approx t_2 && \iff t_1\theta =_E t_2\theta\\
&(tr,\theta) \vDash \neg\varphi && \iff \textit{not } (tr,\theta) \vDash \varphi\\
&(tr,\theta) \vDash \varphi_1 \wedge \varphi_2 && \iff (tr,\theta) \vDash \varphi_1 \textit{ and } (tr,\theta) \vDash \varphi_2\\
&(tr,\theta) \vDash \exists x\colon s.\ \varphi && \iff \textit{there is } u \in \mathrm{dom}(s)\\
& && \quad \textit{such that } (tr, \theta\big[x \mapsto u\big]) \vDash \varphi\,.
\end{aligned}$$

For completeness, we define

$$\begin{aligned}
&\varphi_1 \vee \varphi_2 && \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)\\
&\forall x\colon s.\ \varphi && \equiv \neg(\exists x\colon s.\ \neg\varphi)\\
&\varphi_1 \implies \varphi_2 && \equiv \neg\varphi_1 \vee \varphi_2\\
&\varphi_1 \iff \varphi_2 && \equiv \varphi_1 \implies \varphi_2 \wedge \varphi_2 \implies \varphi_1
\end{aligned}$$

We write $t_1 = t_2$, $i < j$, and $i = j$ when the meaning is clear from context. Timepoints are used to indicate the position of facts in a trace. The free variables of φ are denoted by $fv(\varphi)$ which may be used as a set or sequence depending on the context. We say φ is a *ground formula* if it does not contain free variables, that is, $fv(\varphi) = \emptyset$. When φ is a ground formula, we may write $tr \vDash \varphi$ since the satisfaction of φ is independent of the valuation. The renaming of the free variables of φ by a sequence $\vec{v}$ of equal length is denoted by $\varphi\big[fv(\varphi) \mapsto \vec{v}\big]$ or simply $\varphi\big[\vec{v}\big]$. We write $\varphi\big(\vec{x}\big)$ to denote that the variables $\vec{x}$ are bound in φ, that is, $fv\big(\varphi(\vec{x})\big) = fv(\varphi) \setminus \vec{x}$.

Definition 3 (Validity, satisfiability)**.** *Let Tr be a set of traces. A trace formula φ is* valid *for Tr, written $Tr \vDash^\forall \varphi$, iff $(tr, \theta) \vDash \varphi$ for every trace $tr \in Tr$ and every valuation θ. A trace formula φ is* satisfiable *for Tr, written $Tr \vDash^\exists \varphi$, iff there exists a trace $tr \in Tr$ and a valuation θ such that $(tr, \theta) \vDash \varphi$.*

Note that $Tr \vDash^\forall \varphi$ iff $Tr \nvDash^\exists \neg\varphi$.

h) Instantiations: An *instantiation* ρ is a substitution from variables $\mathcal{V}$ to ground terms $\mathcal{M}$. We say that ρ is *grounding* with respect to a trace formula φ if $\varphi\rho$ is a ground formula. For two instantiations ρ, ρ' we only consider equality modulo E and simply write $\rho = \rho'$. In particular, all operations involving instantiations are considered modulo E.

i) Accountability protocol: The conditions we derive are independent of the formalism of choice. For now, we assume a function *traces* from protocols to sets of ground traces, i.e., sequences of ground facts. Given a protocol P (e.g., a ground process or a set of multiset-rewrite rules), φ is valid for P, written $P \vDash^\forall \varphi$, if $traces(P) \vDash^\forall \varphi$; φ is satisfiable for P, written $P \vDash^\exists \varphi$, if $traces(P) \vDash^\exists \varphi$.

An accountability protocol is a protocol P with a countably infinite set of participants $\mathcal{A}$. We assume that $\mathcal{A} \subseteq \mathcal{M}$, that is, a party can be any ground term. Due to the huge variety in the design of protocols, we leave the concrete structure of this process open. However, we require that each party which is not

trusted specifies a corruption procedure that emits a Corrupted fact and reveals its secrets. The set of corrupted parties of a trace t is defined by

$$corrupted(t) \coloneqq \left\{ A \in \mathcal{A} \;\middle|\; \mathsf{Corrupted}(A) \in t \right\}.$$

In this work, we implicitly assume an accountability protocol P. If not stated otherwise, quantification over traces is always with respect to $traces(P)$.

B. A Definition of Accountability

We review the accountability definition of Künnemann, Esiyok, and Backes [21]. This definition holds parties accountable for violations of a *security property* which is expressed as a trace property φ. To allow any meaningful analysis, there have to be at least two traces, one satisfying and one violating the security property. Following intuition, if all parties adhere to the protocol, the security property φ must hold. Otherwise, either the protocol or φ is ill-defined.

If a violation occurred, i.e., $\neg\varphi$, at least one party must have deviated from the protocol. Each party is either *honest* and follows the protocol or *dishonest* and may deviate from its specified behavior. The definition assumes a single adversary controlling all dishonest parties (see [24] for a discussion of this topic). An honest party becomes dishonest when it receives a *corruption* message from the adversary and remains dishonest for the rest of the protocol execution. We may refer to parties as dishonest or corrupted interchangeably throughout this work.

A dishonest party does not have to deviate and may behave in a way that is indistinguishable from the protocol. It is thus impossible to detect all dishonest parties. Furthermore, parties may deviate by communicating through hidden channels and thus it is also impossible to detect all deviating parties. Instead, Künnemann, Esiyok, and Backes [21] build on sufficient causation [13], [18], and focus on parties that are the actual cause of a violation. This requires protocols to be defined in such a way that deviating parties leave publicly observable evidence for security violations. In this sense, a protocol provides accountability with respect to φ if we can determine all parties *for which the fact that they are deviating at all* is a cause for the violation of φ.

Assume a countably infinite set of parties $\mathcal{A}$.[2] Deviations of a set of parties $S \subseteq \mathcal{A}$ are a cause for a violation iff

SC1: A violation occurred and the parties in S deviated.
SC2: If all deviating parties, except those in S, behaved honestly, the same violation would still occur.
SC3: S is minimal; SC1 and SC2 hold for no strict subset of S.

SC1 ensures that a violation has occurred and the parties in S deviated. SC2 ensures that the parties in S are sufficient to cause a violation. There may be other deviating parties not in S, but their deviation has no influence on the violation. In this vein, SC2 describes a situation which differs from the actual observed events—called a *counterfactual*. SC3 ensures that only minimal sets S are considered, that is, we always hold the least number of parties accountable.

Example 1. Consider a protocol in which access to a central user database is logged and each request must be signed. A violation occurs whenever user data is leaked. The parties involved are a manager M and two employees E_1 and E_2. The manager can directly sign a request to get access to the database and can thus cause a violation on its own. For the employees to gain access, both need to sign a request. Assume this is the case and a leak occurs. Then E_1 and E_2 are *jointly* accountable. In the counterfactual scenarios in which only E_1 or E_2 deviates, a violation is not possible and thus SC2 is not satisfied.

If parties M and E_1 cause a violation, then SC1 and SC2 hold. However, as M can cause a violation on its own, which also satisfies SC1 and SC2, SC3 does not hold. ◊

The counterfactual situations considered in SC2 cannot be chosen arbitrarily. They have to be related to the actual situation to obtain meaningful and justifiable results. This relationship is specified by a *counterfactual relation* r. If $(t, t') \in r$, also written as $r(t, t')$, then the *counterfactual trace* t' is related to the *actual trace* t.

We only consider counterfactual traces if they do not consider additional parties as corrupted, as these are not being causally relevant for a security violation in the actual trace.

Definition 4 (Counterfactual relation)**.** *A counterfactual relation is a reflexive and transitive relation between traces s.t.:*

$$r(t, t') \implies corrupted(t') \subseteq corrupted(t) \tag{1}$$

The *a posteriori verdict* (apv) specifies for a given trace all minimal subsets of parties that are sufficient to cause a security violation.

Definition 5 (A posteriori verdict)**.** *Let P be a protocol, t a trace, φ a security property, and r a relation on traces. The a posteriori verdict is defined by* $apv_{P,\varphi,r}(t) \coloneqq$

$$\Big\{ S \;\Big|\; t \vDash \neg\varphi \tag{R1}$$
$$\wedge\, \exists t'.\, r(t, t') \wedge corrupted(t') = S \wedge t' \vDash \neg\varphi \tag{R2}$$
$$\wedge\, \nexists t''.\, r(t, t'') \wedge corrupted(t'') \subsetneq S \wedge t'' \vDash \neg\varphi \Big\} \tag{R3}$$

We may leave out any of the subscripts P, φ, r if they can be inferred from context. The output of the apv is called a *verdict*.

Example 2. In the situation of Example 1, the following verdicts may be returned by the apv:

$\varnothing$	The empty verdict—no violation and no parties to blame.
$\{\{M\}\}$	The manager leaked the data on its own.
$\{\{E_1, E_2\}\}$	The employees colluded to leak the data.
$\{\{M\}, \{E_1, E_2\}\}$	The manager as well as the employees leaked the data. ◊

Each set $S \in apv_{P,\varphi,r}(t)$ satisfies SC1, SC2, and SC3. R1 ensures that a violation occurred and therefore at least

[2] In contrast to [21] where a finite set of parties is assumed.

one party in S deviated in t. If not all parties in S would deviate, there would be a counterfactual trace t'', where a strict subset of S would deviate, thereby violating R3. Hence, SC1 is satisfied. SC2 is captured by R2, which ensures that there exists a counterfactual trace t', showing that the parties in S are sufficient to cause a violation. SC3 follows directly from R3.

The following corollary shows that accountability with respect to φ implies verifiability of φ. If no violation occurred, no parties are blamed. If no parties are blamed, no violation occurred.

Corollary 6. *For all traces t, $apv_{P,\varphi,r}(t) = \varnothing \iff t \vDash \varphi$.*

Proof. Assume $t \vDash \varphi$. Then $apv_{P,\varphi,r}(t) = \varnothing$ follows by Definition 5. For the other direction, assume $t \vDash \neg\varphi$. As r is reflexive, R2 holds for t and $S = corrupted(t)$. If S is already minimal, there does not exist a trace t'' which corrupts a strict subset of S and thus $apv_{P,\varphi,r}(t) = \{S\} \neq \varnothing$. If S is not minimal, there exists a trace t'' which corrupts $S' \subsetneq S$. The counterfactual trace t' can then be instantiated with t'' and S'. If S' is not minimal, this step can be repeated until a minimal set $S^\star$ is obtained. As the cardinality of the sets decreases in each step, this approach is guaranteed to terminate. ∎

The apv can only be computed after the fact, that is, it requires full knowledge of the actual trace t. The task of an accountability protocol is to always compute the apv without this information. For generality, we assume that an accountability protocol comes with a total function that extracts this information, the *verdict function*:

$$verdict(t) : traces(P) \to 2^{2^{\mathcal{A}}}. \tag{2}$$

Accountability is now defined in terms of the apv and the verdict function. If the apv coincides for all traces with the verdict function, the latter provides the protocol with accountability for a security property φ.

Definition 7 (Accountability)**.** *A verdict function* verdict *provides a protocol P with accountability for a security property φ with respect to a relation r, if*

$$\forall t \in traces(P).\ verdict(t) = apv_{P,\varphi,r}(t). \qquad (\mathsf{Acc}^{verdict}_{P,\varphi,r})$$

Remark 1 (Counterfactual relation)**.** Künnemann, Esiyok, and Backes [21] note that there is no consensus in the causality literature about how actual and counterfactual scenarios should relate. They propose three approaches for relating actual and counterfactual traces: By control flow, by kind of violation, and the weakest relation with respect to (1). Our focus will be on relating traces with the same kind of violations. As we will discuss in Section VI, our method may also be used to encode other relations. The axiomatic characterization in the next section, however, is independent of the choice of r.

IV. Axiomatic Characterization

Accountability (Definition 7) requires that the apv coincides with a given verdict function for all traces of the protocol. In this section, we reformulate this requirement into five equivalent *verification conditions* that are sound and complete. Soundness allows us to prove that a verdict function provides a protocol with accountability by verifying that all conditions hold. Completeness ensures that if a verdict function provides a protocol with accountability, then all conditions hold.

This axiomatic characterization bears resemblance to the verification conditions presented by Künnemann, Esiyok, and Backes [21], but is more general. It is valid for any counterfactual relation and any verdict function. It is also simpler.[3] These axioms will help us derive verification conditions for unbounded sets of participants in the next section in a systematic manner.

A verdict function *verdict* providing accountability for φ and r is characterized by the following axioms.

Verifiability (V_φ) This follows directly from Corollary 6.

$$\forall t.\ verdict(t) = \varnothing \iff t \vDash \varphi$$

We require that whenever the verdict function returns an empty verdict, the security property holds.

Minimality (M_S) This follows directly from Corollary 13.

$$\forall t.\ S \in verdict(t) \implies \nexists S'.\ S' \in verdict(t) \wedge S' \subsetneq S$$

We require that the verdict does not contain a strict subset of one of its sets. Intuitively, this axiom ensures that we only blame the least number of parties which caused a violation.

Sufficiency (SF_S) This axiom is similar to R2 and guarantees that each set of parties in a verdict is sufficient to cause a violation on their own.

$$\forall t.\ S \in verdict(t) \implies \exists t'.\ verdict(t') = \{S\} \wedge corrupted(t') \subseteq S \wedge r(t,t')$$

For each set of parties in a verdict, there exists a related trace in which only a subset of these parties has been corrupted and for which the verdict function returns a singleton verdict only blaming these parties.

We could also define a slightly stronger sufficiency condition, where we would require equality between the set of corrupted parties and the set in the verdict, that is, $corrupted(t') = S$. Instead, we capture this requirement in its own condition—uniqueness.[4] With this approach, we get more precise information when a condition does not hold.

[3] More precisely, the present conditions (P) differ from the coarse-grained conditions (C) [21, Section III] and the fine-grained condition (F) [21, Section IV] as follows. The completeness condition in (F) was found to be incompatible with the definition of the apv. Completeness in (P) is necessarily weaker and in line with the requirements of the other conditions (sufficiency, minimality, uniqueness). Verification is the same in all three. Sufficiency in (P) is slightly weaker than sufficiency in (C) and (F), as it allows for the witness trace to corrupt a subset of the blamed parties. Additionally, sufficiency in (P) requires no violation, but this requirement is superfluous, as it follows from verifiability. Sufficiency for composite verdicts in (F) follows from sufficiency in (P). Uniqueness in (P) and uniqueness for singletons in (F) are the same. Uniqueness in (C) is logically equivalent, but expressed differently. Minimality in (P) is weaker than in (C). Minimality for composite verdicts in (F) can be considered equivalent, but for singleton verdicts it vanishes, because it follows from uniqueness in (F).

[4] The name goes back to the case distinction used in the verdict function of Künnemann, Esiyok, and Backes [21], in which the condition ensures that only a unique, sufficient, and minimal verdict exists for each case.

Uniqueness (U_S) This condition guarantees that all parties in a verdict have been corrupted; or in other words, no honest parties are blamed for a security violation.

$$\forall t.\ S \in \mathit{verdict}(t) \implies S \subseteq \mathit{corrupted}(t)$$

The previous four conditions state the requirements that a group of parties in the verdict must satisfy. The next condition ensures that indeed all groups of parties which satisfy these requirements are included in the verdict.

Completeness (C_S)

$$\forall t.\ \left[\begin{array}{l}\exists t'.\ \mathit{verdict}(t') = \{S\} \\ \quad \wedge\ \mathit{corrupted}(t') \subseteq S \wedge r(t,t')\end{array}\right] \tag{C1}$$

$$\wedge\ \left(\nexists S'.\ S' \in \mathit{verdict}(t) \wedge S' \subsetneq S\right) \tag{C2}$$

$$\wedge\ S \subseteq \mathit{corrupted}(t) \tag{C3}$$

$$\wedge\ t \vDash \neg\varphi \implies S \in \mathit{verdict}(t) \tag{C4}$$

We write SF, V_φ, M, U, and C if the respective condition holds for all S. We denote the conjunction of these conditions by VC_φ or by VC if the security property can be inferred from context.

Theorem 8. *For any protocol P, security property φ, and verdict function* verdict, verdict *provides P with accountability for φ iff* VC.

Proof. In Appendix A, we show soundness and completeness in two separate theorems. ∎

V. Verdict Functions for Unbounded Sets of Participants

The structure of verdict functions proposed by Künnemann, Esiyok, and Backes [21] considers an explicit mapping from observations, i.e., sets of traces, to verdicts. All parties that can occur in a verdict are thus fixed a priori. This prohibits the analysis of several protocol instances in parallel and is inadequate for protocols such as TLS, where a single responder may react to incoming requests from many clients.

If we allow for multiple protocol sessions and consider the set of parties that participate to be unbounded, then, for some protocols, we cannot bound the number of possible verdicts or their size. Therefore, we must define the verdict function indirectly. To this end, we lift the *accountability tests* of Bruni, Giustolisi, and Schuermann [17], which determine whether a given party is to blame, to *case tests*, which can contain variables instead of concrete parties. Case tests are trace properties with free variables. Each free variable is instantiated with a party that should be blamed for a violation. A case test ought to have at least one free variable and there should be at least one trace where it applies.

Definition 9 (Case test)**.** *A case test τ is a trace property which satisfies*

(a) $|\mathit{fv}(\tau)| \geqslant 1$ *and*
(b) $\exists t, \rho.\ t \vDash \tau\rho$.

We say a case test τ *matches* a trace t if there exists an instantiation ρ such that $t \vDash \tau\rho$. The verdict function is now given as the union of all matches.

Definition 10 (Verdict function)**.** *Let $\mathcal{C}$ be a set of case tests. The verdict function induced by $\mathcal{C}$ is given by*

$$\mathit{verdict}_{\mathcal{C}}(t) \coloneqq \bigcup_{\tau\in\mathcal{C}} \left\{ \mathit{fv}(\tau)\rho \;\middle|\; \exists\rho.\ t \vDash \tau\rho \right\}, \tag{3}$$

where the union is modulo the equational theory E.

In the following, we assume a fixed set of user-defined case tests which are denoted by $\mathcal{C} = \tau_1, \ldots, \tau_n$ and write $\mathit{verdict}(t)$.

Example 3. Consider the protocol described in Example 1 in the multi-session setting, that is, there may be multiple managers, employees, and data leaks. There are two possibilities, how a data leak can arise. Either by a manager or by two colluding employees. We want to hold all groups of parties accountable which are responsible for a leak. In contrast to the single-session setting, the protocol must now provide evidence which group of parties leaked the data. Only knowing the parties which accessed the data is not sufficient to identify the parties responsible for a violation. In the case of a single violation, we would suspect all groups of parties that accessed the data.

The security property indicates that neither a manager nor employees leaked the data.

$$\varphi \coloneqq \nexists m, e_i, e_j, \mathit{data}, i.\ \mathsf{LeakManager}(m, \mathit{data})@i \vee \mathsf{LeakEmployees}(e_i, e_j, \mathit{data})@i$$

We define the following two case tests.

$$\tau_1 \coloneqq \exists\, \mathit{data}, i.\ \mathsf{LeakManager}(m, \mathit{data})@i$$
$$\tau_2 \coloneqq \exists\, \mathit{data}, i.\ \mathsf{LeakEmployees}(e_i, e_j, \mathit{data})@i$$

We note that the identities of the manager (m) and employees (e_i, e_j) are free in τ_1 and τ_2 respectively. Given a trace t where two managers M_1, M_2 and each pair of employees E_1, E_2, E_3 caused a violation, the following instantiations exist for τ_1 and τ_2.

$$\rho_1^{(1)} = [m \mapsto M_1] \qquad \rho_2^{(1)} = [e_i \mapsto E_1, e_j \mapsto E_2]$$
$$\rho_1^{(2)} = [m \mapsto M_2] \qquad \rho_2^{(2)} = [e_i \mapsto E_2, e_j \mapsto E_3]$$
$$\rho_2^{(3)} = [e_i \mapsto E_1, e_j \mapsto E_3]$$

We obtain all singleton verdicts of the trace by applying the instantiations to the free variables of the case tests. Hence, the complete verdict is

$$\mathit{verdict}(t) = \{\{M_1\}, \{M_2\}, \{E_1, E_2\}, \{E_2, E_3\}, \{E_1, E_3\}\}.$$ ◊

This example also illustrates that case tests are well suited to distinguish different kinds of a violation, which are identified by the test and its instantiation. We can formalize this notion by assigning each trace the set of case tests with their

corresponding satisfying instantiations.

$$\Lambda(t) \coloneqq \bigcup_{i\in[n]} \{(\tau_i, \rho) \mid \exists \rho.\ t \vDash \tau_i\rho\} \tag{4}$$

We call a trace t *single-matched* if $|\Lambda(t)| = 1$ and *multi-matched* if $|\Lambda(t)| > 1$.

Example 4. In the situation of Example 3, we obtain

$$\Lambda(t) = \{(\tau_1, \rho_1^{(1)}), (\tau_1, \rho_1^{(2)}), (\tau_2, \rho_2^{(1)}), (\tau_2, \rho_2^{(2)}), (\tau_2, \rho_2^{(3)})\}\,. \qquad \Diamond$$

We use the following corollary to justify switching between the verdict-based notation of Section IV and the notation based on case tests of this section.

Corollary 11. *Definition 10 implies that for all traces t*

$$S \in \mathit{verdict}(t) \iff \exists i, \rho.\ t \vDash \tau_i\rho \wedge \mathit{fv}(\tau_i)\rho = S\,.$$

We see that $\Lambda(t)$ contains all the information to compute the verdict for the trace t. Definition 10 implies

$$\Lambda(t') \subseteq \Lambda(t) \implies \mathit{verdict}(t') \subseteq \mathit{verdict}(t)\,. \tag{5}$$

However, Λ provides a more precise picture, since the same set in the verdict may be produced by multiple case tests and instantiations.

We can now instantiate the verification conditions from Section IV with case tests. If $\mathcal{C}$ is finite, we obtain a finite set of conditions, all of which (except sufficiency) are predicates on traces, but not yet trace formulas according to Definition 1. We first apply Corollary 11 to each occurrence of $S \in \mathit{verdict}(t)$ and $\mathit{verdict}(t) = \{S\}$ in the conditions. Since the original conditions are parameterized by S, the resulting conditions are parameterized by a case test τ_i and an instantiation ρ. We reparameterize these conditions with a case test τ_i by introducing quantifiers for the instantiations. As the set of case tests is finite, we also replace quantification over case tests by conjunctions and disjunctions. For an instantiation ρ, we have

$$\exists i.\ t \vDash \tau_i\rho \iff \bigvee_{i\in[n]} t \vDash \tau_i\rho\,, \tag{6}$$

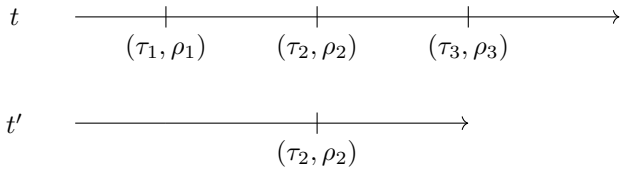
and

$$\forall i.\ t \vDash \tau_i\rho \iff \bigwedge_{i\in[n]} t \vDash \tau_i\rho\,. \tag{7}$$

Moreover, we split the equivalence in the verifiability condition V_φ. This step is not a technical requirement, but we may gain more insight in case a condition does not hold. Finally, we obtain the following intermediate representation.

Sufficiency ($\mathsf{SF}^{\mathrm{in}}_{\tau_i}$) Assume a case test matches a trace t. Then there exists a related trace t' in which only the instantiated parties are corrupted. Moreover, if multiple case tests match, all sets of instantiated parties have to be the same. This ensures

Figure 1. Example: We consider t' a valid counterfactual for t.



that the verdict is a singleton.

$$\begin{aligned}&\forall t, \rho.\ t \vDash \tau_i\rho \implies\\ &\quad \exists t'.\ \Big[\bigwedge_{j\in[n]} \forall \rho'.\ t' \vDash \tau_j\rho' \implies \mathit{fv}(\tau_i)\rho = \mathit{fv}(\tau_j)\rho'\Big]\\ &\qquad \wedge \mathit{corrupted}(t') \subseteq \mathit{fv}(\tau_i)\rho\\ &\qquad \wedge r(t, t')\end{aligned}$$

Verifiability Empty ($\mathsf{VE}^{\mathrm{in}}_{\varphi}$) If there is no case test that matches, then the security property holds. This ensures that the security property can only be violated in the ways described by the case tests.

$$\forall t.\ \Big[\bigwedge_{i\in[n]} \nexists \rho.\ t \vDash \tau_i\rho\Big] \implies t \vDash \varphi$$

Verifiability Nonempty ($\mathsf{VNE}^{\mathrm{in}}_{\varphi,\tau_i}$) The condition requires that if a case test matches, then the security property is violated. This ensures that each case test describes a way to violate the security property.

$$\forall t, \rho.\ t \vDash \tau_i\rho \implies t \vDash \neg\varphi$$

Minimality ($\mathsf{M}^{\mathrm{in}}_{\tau_i}$) The condition ensures that, when a case test matches, then no other case test matches with a strict subset of the instantiated parties.

$$\forall t, \rho.\ t \vDash \tau_i\rho \implies \bigwedge_{j\in[n]} \nexists \rho'.\ t \vDash \tau_j\rho' \wedge \mathit{fv}(\tau_j)\rho' \subsetneq \mathit{fv}(\tau_i)\rho$$

Uniqueness ($\mathsf{U}^{\mathrm{in}}_{\tau_i}$) The condition requires that the instantiated parties of a case test have been corrupted. This ensures that we do not blame honest parties for a security violation.

$$\forall t, \rho.\ t \vDash \tau_i\rho \implies \mathit{fv}(\tau_i)\rho \subseteq \mathit{corrupted}(t)$$

Remark 2. The completeness condition C_S does not need to be encoded as a trace property. We show in Lemma 21 that it follows from $\mathsf{VNE}^{\mathrm{in}}_{\varphi,\tau_i}$, (5), and RE, a requirement on the counterfactual relation we introduce in the next section.

VI. Counterfactual Relation

As we consider an unbounded number of sessions, we can, in general, expect to have multiple causally independent security violations in the same trace. Consider t in Figure 1, where three case tests $\tau_1, \ldots, \tau_3$ match. By VNE_φ, each match implies a security violation by itself. For our counterfactual analysis, we want to consider traces that contain only one of these matches causally relevant. We require the counterfactual relation to be compatible with this intuition, which we formalize

as follows. Note that neither condition restricts the relation for non-violating traces. Indeed, the relation is irrelevant for the apv of those (see Definition 5).

Relation Introduction (RI) For all $i \in [n]$, traces t, t' and instantiations ρ

$$\begin{aligned} & t \vDash \tau_i\rho \\ \wedge\ & \Lambda(t') = \{(\tau_i, \rho)\} \\ \wedge\ & \mathit{corrupted}(t') = \mathit{fv}(\tau_i)\rho \subseteq \mathit{corrupted}(t) \\ \implies\ & r(t, t')\,. \end{aligned}$$

If there is at least one match in some trace t and we can identify a trace t' with the exact same match, corrupting no more parties than t and exactly those indicated by the match, then we consider t' a relevant counterfactual.

Relation Elimination (RE) For all traces t, t'

$$r(t, t') \wedge t \vDash \neg\varphi \wedge t' \vDash \neg\varphi \implies \Lambda(t') \subseteq \Lambda(t)$$

Intuitively, if t' is a relevant counterfactual for t, then it cannot have *additional* matches.

Künnemann, Esiyok, and Backes [21] discuss the lack of a consensus in the causality literature about how actual and counterfactual scenarios should relate. They consider three frequently used relations: r_k, where two traces relate if they have a "similar kind" of violation; r_c, where they need to have the same control flow; and r_w, which is the weakest possible relation, where $r_w(t, t') \iff \mathit{corrupted}(t') \subseteq \mathit{corrupted}(t)$. Neither gives an indication of how to deal with several parallel infractions in the same trace, but they give us a framework to discuss the present proposal.

Considering the kind of violation, r_k provides the most promising interpretation of case tests. This notion originates from criminal law and is used to solve causal problems with the classical "what-if" by considering the event in question in greater detail, e.g., by distinguishing death from shooting from death from poisoning ([2, p. 188]; see also [1, p. 46]). As such, this notion is informal and depends on intuition. Using case tests, we could formalize r_k using r_Λ with $r_\Lambda(t, t')$ iff

$$\emptyset \neq \Lambda(t') \subseteq \Lambda(t) \wedge \mathit{corrupted}(t') \subseteq \mathit{corrupted}(t)\,.$$

The counterfactual has a subset of the matches of the actual, but at least one. r_Λ is consistent with both RE and RI.

While r_k is only informally defined, it is usually straightforward to apply it to a given protocol and a set of case tests. Case by case, we can thus confirm that r_Λ is a formalization of r_k. Each test and possible instance should mark a different "kind" of violation.

By contrast, a direct adoption of the control flow aware relation r_c [10], [13], [20] would not allow for holding all involved parties responsible, as the control flow of t' (Figure 1) is clearly different from t. If we relax the relation to accept a counterfactual trace if its control flow is a prefix of the actual control flow, we would only collect the parties involved in the *first* violation, which is not our goal. We should thus consider only the control flow per session, and allow the order of sessions to be changed.[5] This could be encoded by splitting the case tests, so that each test applies only to a single per-session control flow.

The weakest possible counterfactual relation r_w can not guarantee RE as a related trace could match a completely different case test.

VII. Verification Conditions as Trace Properties

In this section, we bring the axioms from Section IV, which we instantiated with case tests in the last section, into a form that can be verified in an automated way. The most challenging among these is sufficiency.

A. Subset relations

To encode minimality, sufficiency, and uniqueness, we need to express the subset operator and the function *corrupted* in terms of protocol actions. Let $\vec{a} = (a_1, \ldots, a_m)$ and $\vec{b} = (b_1, \ldots, b_n)$. The strict subset operator in $\mathsf{M}^{\mathrm{in}}_{\tau_i}$ can be expressed by

$$[\![\vec{a} \subsetneq \vec{b}]\!] := \Big[\bigwedge_{i\in[m]} \bigvee_{j\in[n]} a_i = b_j\Big] \wedge \Big[\bigvee_{j\in[n]} \bigwedge_{i\in[m]} b_j \neq a_i\Big]\,. \tag{8}$$

The corruption of a party A is recorded as an action $\mathsf{Corrupted}(A)@i$ in the trace. Hence, the subsets in $\mathsf{SF}^{\mathrm{in}}_{\tau_i}$ and $\mathsf{U}^{\mathrm{in}}_{\tau_i}$ can be expressed for a trace t by

$$\mathit{corrupted}(t) \subseteq \vec{a} \iff t \vDash [\![\mathsf{Corrupted} \subseteq \vec{a}]\!] \tag{9}$$

$$\vec{a} \subseteq \mathit{corrupted}(t) \iff t \vDash [\![\vec{a} \subseteq \mathsf{Corrupted}]\!] \tag{10}$$

where

$$[\![\mathsf{Corrupted} \subseteq \vec{a}]\!] := \forall x, k.\ \mathsf{Corrupted}(x)@k \implies \bigvee_{i\in[m]} x = a_i$$

$$[\![\vec{a} \subseteq \mathsf{Corrupted}]\!] := \bigwedge_{i\in[m]} \exists k.\ \mathsf{Corrupted}(a_i)@k\,.$$

B. Sufficiency as a trace property

These encodings allow us to express all conditions as trace properties, except one: $\mathsf{SF}^{\mathrm{in}}_{\tau_i}$. It has two particularities. First, it is of the form $\forall t.\ \exists t'.\ \gamma(t, t')$, which classifies it as a hyperproperty [7]. Since hyperproperties are in general more expressive than trace properties, they cannot be directly converted to the latter. Second, it is the only condition that contains the counterfactual relation r.

To derive a trace property, we need to get rid of the outermost universal quantifier and abstract the relation r. To avoid the quantifier, we will focus on single-matched traces, i.e., traces with exactly one violation and introduce three additional conditions. They ensure that there exists a single-matched trace (a) for any case test, (b) for any instance thereof, and (c) that matching assignments are always injective. These properties can be considered well-formedness conditions on the case tests and are automatically verified. They define a class of protocols and case tests for which our trace properties are

[5] As our execution model cannot capture the control flow of deviating parties, this only concerns the trusted parties.

sound and complete.[6] To abstract the relation r, we make use of the assumption introduced in Section VI, which is not automatically verified.

We can express that a trace is single-matched as a trace property SM_{τ_i} (see Table I). For a trace t to be single-matched, i.e., $\Lambda(t) = \{(\tau_i, \rho)\}$, three conditions have to be satisfied. First, τ_i has to match t; second, if it matches multiple times, then all variable assignments have to be equal; and finally, no other case test may match t. We write SM if SM_{τ_i} holds for all $i \in [n]$.

With SM_{τ_i}, (9), and RI, we can express the consequent of $\mathsf{SF}^{\text{in}}_{\tau_i}$ as a trace property

$$\exists t, \rho.\ \Lambda(t) = \{(\tau_i, \rho)\} \wedge \mathit{corrupted}(t) \subseteq \mathit{fv}(\tau_i)\rho \qquad (11)$$

This guarantees the existence of a single trace for each case test, but not for all possible instantiations. We thus need to ensure that if (11) holds for a single instantiation, then it also holds for all possible instantiations. To achieve this, we first introduce an additional requirement on the counterfactual relation, the replacement property RP_{τ_i} (see Table I).

Assuming SM_{τ_i} holds, i.e., some single-matched t' exists, then RP_{τ_i} ensures that for any multi-matched trace t with a match for τ_i and ρ, there is a single-matched trace t'' with the same case test and instantiation as t. The property is slightly more general, as t' could corrupt more parties than necessary. To illustrate this point: If t' corrupts the minimal set of parties, i.e., $\mathit{corrupted}(t') = \mathit{fv}(\tau_i)\rho'$, then $\mathit{corrupted}(t'') = \mathit{fv}(\tau_i)\rho$. We write RP if RP_{τ_i} holds for all $i \in [n]$. In other words, SM and RP ensure that there is a decomposition of each trace that separates interleaving causally relevant events so they can be regarded in isolation. A sufficient criterion is that traces are closed under bijective renaming, which we can ensure syntactically by verifying that no public names appear in the protocol and that Corrupted actions contain only variables of sort public.[7]

To ensure that ${\rho'}^{-1}$ is well defined, we require each free variable to be instantiated with a distinct value. This can be expressed as a trace property, Instance Injectivity II (see Table I). We write II if II_{τ_i} holds for all $i \in [n]$. This condition is w.l.o.g. If a case test violates II, it can be split into multiple case tests for each coincidence of instantiated variables.

Example 5. Assume a case test τ_i with $\mathit{fv}(\tau_i) = \{x, y, z\}$ that violates II_{τ_i} and all free variables coincide in any combination. These are given by the partitions of the free variables.

$$\{\{x, y, z\}\} \qquad \{\{z\}, \{x, y\}\}$$
$$\{\{x\}, \{y, z\}\} \qquad \{\{x\}, \{y\}, \{z\}\}$$
$$\{\{y\}, \{x, z\}\}$$

We then need to split τ_i into five case tests in which the variables in each group are replaced by a single variable. For example, if y and z coincide, we replace each occurrence of them by a new variable $v_{y,z}$. ◊

Injectivity of the instantiations also ensures that the number of instantiated variables corresponds to the number of free variables.

$$\left|\mathit{fv}(\tau_i)\rho\right| = \left|\mathit{fv}(\tau_i)\right|$$

Example 6. Consider the situation of Example 3 and a trace t in which a manager M_1 and the employees E_1, E_2 cause a violation. Assume there exist single-matched traces t_1, t_2 with

$$\Lambda(t_1) = \{(\tau_1, [m \mapsto M_2])\}$$
$$\Lambda(t_2) = \{(\tau_2, [e_i \mapsto E_3, e_j \mapsto E_4])\},$$

where only the necessary parties are corrupted. By RP_{τ_1}, there exists a trace t_1' with

$$\Lambda(t_1') = \{(\tau_1, [m \mapsto M_1])\}$$
$$\mathit{corrupted}(t_1') = \mathit{corrupted}(t_1)[M_2 \mapsto M_1] = \{M_1\}.$$

By RP_{τ_2}, there exists a trace t_2' with

$$\Lambda(t_2') = \{(\tau_2, [e_i \mapsto E_1, e_j \mapsto E_2])\}$$
$$\mathit{corrupted}(t_2') = \mathit{corrupted}(t_2)[E_3 \mapsto E_1, E_4 \mapsto E_2] = \{E_1, E_2\}.$$ ◊

C. Soundness and Completeness

In this section, we defined the class of protocols and case tests where we can express sufficiency as a trace property by stating SM, II and RP. The latter can be checked syntactically, while the first two can be verified directly.

Using the results we obtained above, we can now finally define the verification conditions in terms of trace properties. In the following, we assume P to be an accountability protocol.

We write SF^{tp}, $\mathsf{VE}^{\text{tp}}_{\varphi}$, $\mathsf{VNE}^{\text{tp}}_{\varphi}$, M^{tp}, and U^{tp} if the respective condition holds for all $i \in [n]$. We denote the conjunction of these conditions by $\mathsf{VC}^{\text{tp}}_{\varphi}$ or by VC^{tp} if the security property can be inferred from context.

We show the correctness of these conditions by relating them to the axiomatic characterization from Section IV, which has been proven equivalent to Definition 5 in Theorem 8. Lemma 16 to Lemma 20 in Appendix B and Table I give a nuanced picture of their relationship, which is useful to interpret counterexamples (see also Appendix D). Theorem 22 and Theorem 23 in Appendix B show soundness and completeness.

VIII. Verifying Accountability using Tamarin

Tamarin [12] is a protocol verification tool that supports falsification and unbounded verification in the symbolic model. This makes Tamarin particularly suitable for integrating our results. We extended Tamarin with two syntactical elements, case tests and accountability lemmas. Case tests are specified by

```
test ⟨name⟩:
  "⟨τ⟩"
```

[6] Alternatively, they can be understood as part of the verification conditions. In this case, we offer two sets of conditions, one that is sound and one that is complete.

[7] This simple check applies to both multiset-rewrite rules and SAPiC processes. A more refined syntactic condition is possible by allowing for public names unless they are compared to variables that occur in verdicts. In our case studies, we verified this condition by hand.

Table I
VERIFICATION CONDITIONS

	name	definition	logical relation
sufficiency (tr. prop.)	SM_{τ_i}	$P \vDash^{\exists} \exists \vec{v}.\ \tau_i[\vec{v}] \quad \wedge \Big[\forall \vec{w}.\ \tau_i[\vec{w}] \implies \vec{w} = \vec{v}\Big] \wedge \Big[\bigwedge_{j \in [n]\setminus\{i\}} \nexists \vec{x}.\ \tau_j[\vec{x}]\Big]$	
	II_{τ_i}	$P \vDash^{\forall} \forall \vec{v}.\ \tau_i[\vec{v}] \implies \bigwedge_{i \in idx(\vec{v})} \bigwedge_{\substack{j \in idx(\vec{v}) \\ j \neq i}} v_i \neq v_j$	$\mathsf{SF}^{\mathrm{tp}} \wedge \mathsf{VNE}^{\mathrm{tp}}_{\varphi} \wedge \mathsf{U}^{\mathrm{tp}} \wedge \mathsf{II} \wedge \mathsf{RP} \implies \mathsf{SF}$ (Lemma 19)
	$\mathsf{SF}^{\mathrm{tp}}_{\tau_i}$	$P \vDash^{\exists} \exists \vec{v}.\ \tau_i[\vec{v}] \wedge \Big[\forall \vec{w}.\ \tau_i[\vec{w}] \implies \vec{w} = \vec{v}\Big] \wedge \Big[\bigwedge_{j \in [n]\setminus\{i\}} \nexists \vec{x}.\ \tau_j[\vec{x}]\Big]$ $\wedge \forall a, k.\ \mathsf{Corrupted}(a)@k \implies \bigvee_{\ell \in idx(\vec{v})} a = \vec{v}_\ell$	$\mathsf{SF} \wedge \mathsf{SM} \implies \mathsf{SF}^{\mathrm{tp}}$ (Lemma 20)
other conditions (tr. prop.)	$\mathsf{VE}^{\mathrm{tp}}_{\varphi}$	$P \vDash^{\forall} \Big[\bigwedge_{i \in [n]} \nexists \vec{v}.\ \tau_i[\vec{v}]\Big] \implies \varphi$	$\mathsf{V}_{\varphi} \iff \mathsf{VE}^{\mathrm{tp}}_{\varphi} \wedge \mathsf{VNE}^{\mathrm{tp}}_{\varphi}$ (Lemma 16)
	$\mathsf{VNE}^{\mathrm{tp}}_{\varphi,\tau_i}$	$P \vDash^{\forall} \forall \vec{v}.\ \tau_i[\vec{v}] \implies \neg\varphi$	
	$\mathsf{M}^{\mathrm{tp}}_{\tau_i}$	$P \vDash^{\forall} \forall \vec{v}.\ \tau_i[\vec{v}] \implies \bigwedge_{j \in [n]} \nexists \vec{w}.\ \tau_j[\vec{w}] \wedge [\![\vec{w} \subsetneq \vec{v}]\!]$	$\mathsf{M} \iff \mathsf{M}^{\mathrm{tp}}$ (Lemma 17)
	$\mathsf{U}^{\mathrm{tp}}_{\tau_i}$	$P \vDash^{\forall} \forall \vec{v}.\ \tau_i[\vec{v}] \implies \bigwedge_{\ell \in idx(\vec{v})} \exists k.\ \mathsf{Corrupted}(\vec{v}_\ell)@k$	$\mathsf{U} \iff \mathsf{U}^{\mathrm{tp}}$ (Lemma 18)
syntactic	RP_{τ_i}	$\forall t, t', \rho, \rho'.\ t \vDash \tau_i\rho \wedge \Lambda(t') = \{(\tau_i, \rho')\}$ $\implies \exists t''.\ \Lambda(t'') = \{(\tau_i, \rho)\} \wedge corrupted(t'') = corrupted(t')(\rho \circ \rho'^{-1})$	$\mathsf{BR} \implies \mathsf{RP}$
	BR	$\forall t, \sigma\colon \mathcal{A} \leftrightarrow \mathcal{A}.\ t \in traces(P) \implies t\sigma \in traces(P)$	$\mathcal{A} \subseteq PN \wedge fn(P) \cap PN = \emptyset \implies \mathsf{BR}$ (Lemma 24)

where ⟨name⟩ is the name of the case test and $\langle\tau\rangle$ is its formula. Accountability lemmas are defined similarly to standard lemmas

lemma ⟨name⟩:
⟨name$_1$⟩,...,⟨name$_n$⟩ **account(s) for** "$\langle\varphi\rangle$"

where ⟨name⟩ is the name of the lemma, ⟨name$_i$⟩ are the names of previously defined case tests, and $\langle\varphi\rangle$ is the security property. The implementation allows defining an arbitrary number of accountability lemmas. This is especially useful when experimenting with different sets of case tests, discovering potential attacks, and analyzing accountability properties in general. Each accountability lemma consists of a set of case tests and a security property and thus specifies a verdict function according to Definition 10.

We translate each accountability lemma into a set of standard lemmas stating the trace properties $\mathsf{SF}^{\mathrm{tp}}_{\tau_i}$, $\mathsf{VE}^{\mathrm{tp}}_{\varphi}$, $\mathsf{VNE}^{\mathrm{tp}}_{\varphi,\tau_i}$, $\mathsf{M}^{\mathrm{tp}}_{\tau_i}$, $\mathsf{U}^{\mathrm{tp}}_{\tau_i}$, II_{τ_i}, SM_{τ_i}. In the full version [25, Appendix F] we show that all these lemmas adhere to the guardedness requirement of TAMARIN provided that the case tests are guarded. An accountability lemma holds for a protocol P if TAMARIN can successfully verify all generated lemmas and the replacement property RP holds. A protocol can be specified in terms of multiset-rewrite rules or as a SAPiC process.

When analyzing an accountability lemma, two outcomes are possible. Either TAMARIN is able to verify all conditions or at least one condition is violated. In the latter case, it can be difficult to interpret the attack, depending on whether the condition was necessary. To this end, we provide a detailed decision diagram in Appendix D.

IX. CASE STUDIES

We demonstrate our methodology on eight case studies, four from prior work [21] and four more in the domain of mixnets and electronic voting. We summarize our findings in Tables II and III. For each case study, we provide the verification result (✓ for successful verification, ✗ if we found an attack), the number of generated lemmas, and the time needed to verify all lemmas (even if an attack is found).

Before describing the case studies, we want to emphasize the importance of distinguishing between sessions, roles, and parties. The number of sessions specifies how many instances of a protocol can be executed in parallel. In each session, there can be multiple roles—for example, a server or a client—with different frequencies. Within a protocol trace, these roles are instantiated with concrete party identifiers drawn from a countably infinite set of public names. Depending on the protocol, a party may participate in multiple sessions and each session may be run by different sets of parties. Even if a protocol has just one role, an unbounded number of parties may be involved.

Table II
VERIFICATION RESULTS FOR THE DMN AND MIXVOTE CASE STUDIES IN TWO FRAMEWORKS. WE COMPARE TYPE OF ATTACK (✗=ATTACK,✓=VERIFICATION), NUMBER OF LEMMAS AND OVERALL VERIFICATION TIME.

Our proposal	1 role	2 roles	3 roles	4 roles	5 roles
Basic DMN (duplicate ciphertexts)	—	—	✓ 13 26 s	—	—
DMN + message tracing (first)	✓ 7 8 s	✓ 7 124 s	✓ 7 1373 s	✓ 7 14 178 s	✓ 7 134 160 s
DMN + message tracing (all)	✓ 7 6 s	✗ 7 12 s	✗ 7 22 s	✗ 7 100 s	✗ 7 355 s
MixVote (unbounded)	✓ 14 6 s	—	—	—	—
[21]	1 party	2 parties	3 parties	4 parties	5 parties
DMN + message tracing (first)	✓ 7 7 s	✓ 17 133 s	✓ 46 2146 s	✓ 149 23 827 s	—* 544 —
DMN + message tracing (all)	✓ 7 4 s	✗ 17 23 s	✗ 46 115 s	✗ 149 548 s	✗ 544 2922 s
MixVote (unbounded)**	✓ 14 5 s	✓ 34 58 s	✓ 92 2721 s	—* 298 —	—* 1112 —

* No verification results due to memory exhaustion. ** Each party acts in the same role, that of the server.

Table III
VERIFICATION RESULTS FOR CASE STUDIES FROM [21] IN THE UNBOUNDED SETTING.

	Our proposal	[21]
WhoDunit (fixed)	✓ 7 52 s	✓ (r_c) 8 24 s
		✓ (r_w) 7 11 s
Certificate Transparency (extended)	✓ 27 17 s	✓ 31 21 s
OCSP Stapling (trusted resp.)	✓ 7 1 s	✓ 7 515 s
OCSP Stapling (untrusted resp.)	✗ 7 1 s	✗ 7 75 s

A. *Case studies from [21]*

We briefly recall the case studies from prior work [21]. *WhoDunit* illustrates a situation where a third party J cannot provide a correct verdict. S sends some value to A and J and A should forward it to J. We are interested in accountability for J receiving the same value from A and S. Without signatures it is impossible to distinguish between A tampering with the message that it should forward and S sending different values to A and J. The fixed version uses signatures to give evidence of provenance. We extended the fixed version to an unbounded number of parties in the roles of A and S. The original version considers only a single communication session, hence both the analysis with respect to r_c and r_w (see Section VI) run faster, because they need to consider only a very small number of possible interleavings (three protocol messages).

Certificate Transparency [28] is an accountability protocol that provides transparency for a public key infrastructure. Künnemann, Esiyok, and Backes [21] extended a simple model [17] for a single certificate authority and a single logging authority. We adapted the model to allow for an unbounded number of both, but otherwise adhered to their original formulation. We observe a slight speed up in the verification, which is likely due to the removal of logical redundancies in the axiomatic characterization shown in Section IV. In contrast to WhoDunit, the original model already considered an unbounded number of interactions between concrete parties. Hence the proofs are similarly structured.

OCSP Stapling [26] is a mechanism to attach signed Online Certificate Status Protocol (OCSP [27]) messages during a TLS handshake. The server's goal is to provide evidence that their certificate has not been revoked recently without the client exposing their browsing behavior to the OCSP server. The model from Künnemann, Esiyok, and Backes [21] used an explicit clock process to model time. We extended their model to an unbounded number of clients, TLS servers, and OCSP servers. Moreover, we ported their SAPiC model to multiset-rewrite rules to exploit a more effective modeling of timepoints, improving the verification time by two orders of magnitude. Otherwise, in particular concerning the communication, we remained faithful to their modeling. The new model of timepoints avoids the use of helping lemmas compared to three required previously. It also reduces the verification time by two orders of magnitude in the case the OCSP responder is trusted and accountability holds and by at least one in case the OCSP responder is untrusted.

B. *Mixnets*

Mixnets are a building block for many privacy-preserving technologies, e.g., e-voting systems, anonymous messaging, anonymous routing, and oblivious RAM (see a recent survey [23]). While basic mixnets are only suitable in the honest-but-curious attacker model, they can be extended to provide verifiability and even accountability.

In this work, we focus on basic decryption mix nets (basic DMN) and their extension with message tracing (DMN +

message tracing) as proposed in [23]. To the best of our knowledge, this case study provides the first automated formal verification results for these kinds of DMNs.

In a DMN, each sender encapsulates their plaintext within several layers of encryption using the last mix server's public key first and the first mix server's public key last. Each mix server decrypts the messages it receives, removing the outermost layer. Each mix server shuffles the messages before sending them to the next server on a public channel. For accountability, we assume these messages to be stored on a public append-only bulletin board that cannot be tampered with.

In basic DMNs, the ciphertexts on the bulletin board are continuously checked for duplicates and, in the case of a duplicate, the protocol is terminated. Depending on which phase they were posted in, this audit correctly identifies the responsible mix server or sender.

In DMNs with message tracing, the senders store the random coins they used for encryption and each intermediate ciphertext they produce. During the audit, each sender verifies for each mixing step that their intermediate ciphertexts appears on the bulletin board. If this is not the case, the sender in question uses their stored random coins to prove that the mix server misbehaved. We consider two cases of DMNs with message tracing: in the first, the sender stops the audit once the first misbehaving mix server is found. In the second variant, the audit continues until the last mix server.

For this case study, we modeled basic DMNs and two variants of DMNs with message tracing in TAMARIN. For the former, we allow three mix server roles and two sender roles. For the latter, we fix the number of sender roles to two, and scale the number of mix server roles, the *mix length*, from one to five. Note that there is still an unbounded number of sessions with an unbounded number of potential senders and mix servers in these roles, similar to how the Tor network fixes the number of onion routers to three, but has millions of users.

For the basic DMN, we can show accountability for duplicate ciphertexts when we limit the senders and mix servers to only duplicate messages, but not submit otherwise dishonestly generated messages. We define two case tests, one for senders, which checks for duplicates in the senders' output, and one for mix servers, which checks for duplicates in the mix servers' output including the final output. The tests hold any party accountable that posts a ciphertext that has already been posted on the bulletin board. Together, they provide accountability for the property that no duplicates occur in the same phase of a session.

For DMN with message tracing, we define a single case test holding the mix servers accountable that have been identified by a sender during the audit

$$\tau := \exists\, sid, x, i.\ m = \langle sid, x\rangle \wedge \mathsf{B}(m)@i\,, \tag{12}$$

where $\mathsf{B}(\langle sid, m\rangle)$ denotes that a server blamed the mix server m in session sid. We note that the variable m is free in the case test. Hence, the parties in the verdict are pairs consisting of a session identifier and a mix server. This allows a mix server to be honest in one session and dishonest in another.

Up to a mix length of five, we can show accountability when the senders/auditors only blame the first mix server they catch cheating. This confirms an existing formal result in the cryptographic model [16]. For the variant where they blame all mix servers who have not posted the correct intermediate ciphertext on the bulletin board, we find a counterexample up to a mix length of five—with one exception. If there is only one mix server role, this case is equivalent to the other variant and accountability holds. For a mix length of two or more, we find that uniqueness is violated, indicating that a mix server can be blamed despite acting honestly. This happens when a dishonest mix server tampers with the ciphertext in one of the previous stages, as the mix servers down the line will themselves produce ciphertexts that fail the audit.

For comparison and to evaluate the impact of using case tests instead of defining the verdict function explicitly, we ported the two variants of DMNs with message tracing to the framework of [21]. First, we had to limit the number of sessions to one. Listing all pairs of mix server identities (which include session identifiers) would have been impossible.

Comparing the results of our approach with the results of [21] in Table II, we see that they agree on the outcome. We note, however, that while the number of generated lemmas stays constant with an increasing number of mix servers in our approach, they increase exponentially in the other. This is, again, due to the explicit enumeration of all cases in the verdict function in [21]. Even though we fix the identities to the number of roles, i.e., $M1$ to $M3$ in the case of a mix length of three, we have to account for each combination of mix servers in the verdict function, e.g.,

$$\mathit{verdict}(t) := \begin{cases} \{\{M1\}\} & \text{if } \omega_{M1}(t) \\ \{\{M2\}\} & \text{if } \omega_{M2}(t) \\ \{\{M3\}\} & \text{if } \omega_{M3}(t) \\ \{\{M1\},\{M2\}\} & \text{if } \omega_{M1,M2}(t) \\ \{\{M1\},\{M3\}\} & \text{if } \omega_{M1,M3}(t) \\ \{\{M2\},\{M3\}\} & \text{if } \omega_{M2,M3}(t) \\ \{\{M1\},\{M2\},\{M3\}\} & \text{if } \omega_{M1,M2,M3}(t) \\ \varnothing & \text{otherwise}\,, \end{cases}$$

Here, each ω_S is a trace property that is satisfied if and only if the annotated mix servers in S are blamed. The number of cases in the verdict function equals the cardinality of the powerset of the set of "blameable" parties, which grows exponentially. Hence, scalability is severely limited by this approach.

Thanks to the use of case tests, our approach permits the specification of the verdict function independent of the number of parties and even the number of mix server roles, keeping the user's specification effort minimal. Another consequence is that the accountability lemmas we produce are actually the same. We nevertheless observe an increase in verification time with the mix length, but this is expected, as the backward-resolution approach in TAMARIN has to explore a larger state space. While a smarter encoding of the case study might be possible—we tried several—this effect would likely occur when verifying

other properties, e.g., correspondence of output and input, in the same model. Compared to the previous approach, we see that the verification time is drastically reduced, sometimes by a factor of five. This is despite the restriction to a fixed set of parties and a single session for the previous approach. The difference is more pronounced the longer the mixnet is, which can be explained as follows: When the mix lengths is increased, the search space for the backward-resolution is increased, which, in both approaches, affects the verification time *per lemma*. Since a lot more lemmas need to be proven for the previous approach, the effect is amplified by a factor that increases with the mix length.

C. Dispute resolution in MixVote

MixVote [22] adds a dispute resolution procedure to the mixnet-based voting protocol Alethea [19].

We first give a high-level overview of the protocol. A voter H uses their device D to compute their ballot by first encrypting their vote under the voting authority's (server S) public key which is then signed by D. The voter casts their ballot b by submitting it to some platform P, which forwards it over the network to S. The ballot is verified by S by checking whether it contains a signature corresponding to an eligible voter who has not previously voted. In this case, b is added to the list of recorded ballots $[b]$. S then signs b and sends it back to H as an evidence that b was indeed received by the authority. This evidence is kept by H in the case for later disputes. Once the voting phase is over, S computes the tally of the recorded ballots $[b]$ by decrypting them using a mixnet. Finally, S publishes the encrypted ballots and the decrypted votes on the public bulleting board such that a voter can verify that their ballot is included.

This case study shows that our approach can be applied to existing specifications with minimal effort and is based on one of the TAMARIN models from Basin, Radomirovic, and Schmid [22] (mixvote_SmHh). In this model, the voting authority S can be corrupted while the voters are honest. When corrupted, the authority's secret key is given to the adversary and the incoming and outgoing channels are modeled as insecure. S is partially trusted to sign and return a valid ballot received from P. This model covers the case where a voter H claims to have cast a ballot b while the authority S claims that this is not the case.

The original model runs a single session of the protocol with the identity of the server fixed to '`S`'. We extended the model to support an unbounded number of sessions, used an unreliable insecure channel from P to S, and added a corruption mechanism for the server. Note that the restriction to a single server role per session is a property of the protocol and not a limitation of our approach.

We focus on the two properties protecting an honest voter in the case of a dispute:

- $VoterC$: ensures that whenever an honest voter detects that one of their ballots was not recorded correctly, they can convince others that S is dishonest.
- $TimelyP$ ensures that whenever an honest voter casts a ballot, they cannot be prevented from continuing the protocol until their ballot is recorded or they can convince others that S is dishonest.

We define an accountability lemma for each property.

a) Accountability for $VoterC$: We first define the security property, which is directly encoded in $VoterC$. Whenever an honest voter H validates their ballot, it is indeed included in the list of recorded ballots on the bulletin board.

$$\varphi_{VoterC} \coloneqq \forall H, b, b1, i.\ \mathsf{Verify}(H, b, b1)@i \implies \exists BB, j.\ \mathsf{BB}_{rec}(BB, \langle \text{`b'}, b\rangle)@j$$

We then define a case test which blames S whenever a ballot is recorded on the bulletin board that has not been signed by the voter's device D and the verifiability check is reached.

b) Accountability for $TimelyP$: The security property follows with a slight change from $TimelyP$. Whenever an honest voter H casts a ballot and all relevant information is published on the bulletin board, the ballot is indeed included in the list of recorded ballots on the bulletin board.

$$\varphi_{TimelyP} \coloneqq \forall H, b, i, j.\ \mathsf{Ballot}(H, b)@i \wedge \mathsf{End}@j \implies \exists BB, k.\ \mathsf{BB}_{rec}(BB, \langle \text{`b'}, b\rangle)@k \wedge i < k \wedge k = j$$

We define a case test which blames S whenever a ballot is recorded on the bulletin board which has not been signed by the voter's device D and the point where the voter receives their ballot from D is reached. We note that this case test is the same as the one for $VoterC$ with the exception of the point in the protocol needed to be reached. Here, we have to slightly strengthen the accountability lemma compared to $TimelyP$. We move the requirement that the ballot is cast before the voting ends ($i < k$) from the premise to the conclusion. Otherwise, when a ballot is cast after the vote has ended, we have a matching case test without a security violation, i.e., a counterexample to $\mathsf{VNE}^{\mathrm{tp}}$.

Our approach can automatically show that accountability holds for the two properties described above without requiring helping lemmas.

We ported the model to the framework of [21] to provide a comparison with our approach. This version also supports an unbounded number of sessions, but due to the restriction on concrete party identifiers, we had to limit the set of parties that could act as the server. We analyzed the protocol with up to five distinct server parties and obtained results with up to three. In the case of four and five identities, TAMARIN's search algorithm exceeded the amount of available memory.

The results in the framework of [21] agree with the results of our approach, but the time needed to obtain them increases exponentially with the number of parties, whereas the result presented here holds for an unbounded number of parties.

For future work, it might be interesting to hold both the voter and server accountable at the same time, by merging the MixVote model that covers a dishonest voter and an honest authority (mixvote_ShHm) with the one we investigated here.

X. Conclusion

In this work, we provide an automated verification methodology for accountability that supports an unbounded number of participants, and thus an unbounded number of security violations. This precludes explicit assignment of blame. We therefore introduced case tests—a higher-level variant of Bruni, Giustolisi, and Schuermann's accountability tests— and used them to define highly flexible verdict functions. Our approach also improves readability, as we may consider each case test as a specific manifestation of a violation. We showed how the verdict-based verification conditions can be expressed using case tests and finally be encoded in terms of trace properties. Furthermore, we extended Tamarin with the ability to automatically generate these from accountability lemmas. Our case studies demonstrate applications for transparency protocols, revocation protocols, mixnets, and dispute resolution in e-voting.

Acknowledgements: This research was partly supported by the ERC Synergy Grant "imPACT" (No. 610150).

References

[1] J. L. Mackie, *The Cement of the Universe: A Study of Causation*. Clarendon Press, 1980.

[2] J. Dressler, *Understanding criminal law*. Matthew Bender, 1995.

[3] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *Proceedings. 1998 IEEE Symposium on Security and Privacy*, 1998, pp. 86–99.

[4] M. Backes, J. Camenisch, and D. Sommer, "Anonymous yet accountable access control," in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, ser. WPES '05, Alexandria, VA, USA: ACM, 2005, pp. 40–46.

[5] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 175–188, 2007.

[6] R. Jagadeesan, A. Jeffrey, C. Pitcher, and J. Riely, "Towards a theory of accountability and audit," in *Proceedings of the 14th European Conference on Research in Computer Security*, ser. ESORICS'09, Saint-Malo, France: Springer-Verlag, 2009, pp. 152–167.

[7] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *J. Comput. Secur.*, vol. 18, no. 6, pp. 1157–1210, 2010.

[8] R. Küsters, T. Truderung, and A. Vogt, "Accountability: Definition and relationship to verifiability," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10, Chicago, Illinois, USA: ACM, 2010, pp. 526–535.

[9] J. Feigenbaum, A. D. Jaggard, and R. N. Wright, "Towards a formal model of accountability," in *Proceedings of the 2011 New Security Paradigms Workshop*, ser. NSPW '11, Marin County, California, USA: ACM, 2011, pp. 45–56.

[10] M. Kuntz, F. Leitner-Fischer, and S. Leue, "From probabilistic counterexamples via causality to fault trees," in *Computer Safety, Reliability, and Security*, F. Flammini, S. Bologna, and V. Vittorini, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 71–84.

[11] M. Backes, D. Fiore, and E. Mohammadi, "Privacy-preserving accountable computation," in *Computer Security – ESORICS 2013*, J. Crampton, S. Jajodia, and K. Mayes, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 38–56.

[12] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 696–701.

[13] A. Datta, D. Garg, D. K. Kaynar, D. Sharma, and A. Sinha, "Program actions as actual causes: A building block for accountability," in *Proceedings of the 28th Computer Security Foundations Symposium*, 2015.

[14] G. Gössler and D. Le Métayer, "A general framework for blaming in component-based systems," *Science of Computer Programming*, vol. 113, Part 3, 2015.

[15] J. A. Kroll, "Accountable algorithms," Ph.D. dissertation, Princeton University, 2015.

[16] R. Küsters, J. Müller, E. Scapin, and T. Truderung, "sElect: A Lightweight Verifiable Remote Voting System," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, Jun. 2016, pp. 341–354.

[17] A. Bruni, R. Giustolisi, and C. Schuermann, "Automated analysis of accountability," in *Information Security*, P. Q. Nguyen and J. Zhou, Eds., Cham: Springer International Publishing, 2017, pp. 417–434.

[18] R. Künnemann, *Sufficient and necessary causation are dual*, 2017. arXiv: 1710.09102 [cs.AI].

[19] D. Basin, S. Radomirovic, and L. Schmid, "Alethea: A provably secure random sample voting protocol," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, IEEE, 2018, pp. 283–297.

[20] R. Künnemann, D. Garg, and M. Backes, *Accountability in security protocols*, Cryptology ePrint Archive, Report 2018/127, 2018.

[21] R. Künnemann, I. Esiyok, and M. Backes, "Automated verification of accountability in security protocols," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, 2019, pp. 397–39 716.

[22] D. A. Basin, S. Radomirovic, and L. Schmid, "Dispute resolution in voting," in *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*, IEEE, 2020, pp. 1–16.

[23] T. Haines and J. Muller, "SoK: Techniques for Verifiable Mix Nets," in *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, Boston, MA, USA: IEEE, Jun. 2020, pp. 49–64.

[24] R. Künnemann, D. Garg, and M. Backes, "Accountability in the decentralised-adversary setting," in *2021 IEEE*

34nd Computer Security Foundations Symposium (CSF), 2021.

[25] K. Morio and R. Künnemann, *Verifying accountability for unbounded sets of participants*, Full version, 2021. arXiv: 2006.12047.

[26] D. Eastlake 3rd, *Transport Layer Security (TLS) Extensions: Extension Definitions*, RFC 6066 (Proposed Standard), Fremont, CA, USA, Jan. 2011.

[27] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*, RFC 6960 (Proposed Standard), Fremont, CA, USA, Jun. 2013.

[28] B. Laurie, A. Langley, and E. Kasper, *Certificate Transparency*, RFC 6962 (Experimental), Fremont, CA, USA, Jun. 2013.

APPENDIX

A. Proofs for Section IV

For the proofs see the full version [25, Appendix A].

1) Helping lemmas: The following lemma will help us in the soundness proof. Assume actual and counterfactual traces t, t' are related and S' is a set in $apv(t')$. Then $apv(t)$ is empty or there exists a subset of S which is in $apv(t)$.

Lemma 12. *For all traces t, t'*

$$r(t,t') \land S' \in apv(t') \implies apv(t) = \varnothing \lor \exists S.\, S \in apv(t) \land S \subseteq S'\,.$$

It may seem unintuitive that either the apv is empty or for each set of parties in the apv of the counterfactual trace a subset of this set must exist in the apv of the actual trace. We note that the minimality requirement of the apv is weaker in the counterfactual trace than in the actual trace. The traces related to t' are a subset of the traces related to t and thus there may be a trace related to t showing that a set S is not minimal, but this trace is not necessarily related to t'.

From R3 of the apv, we derive that the apv does not contain two sets where one is a strict subset of the other.

Corollary 13. *For all traces t and sets S*

$$S \in apv(t) \implies \nexists S'.\, S' \in apv(t) \land S' \subsetneq S\,. \tag{13}$$

2) Soundness and Completeness: We show that the verdict-based verification conditions are sound and complete with respect to Definition 7.

Theorem 14 (Soundness)**.** *For any protocol P, security property φ, and verdict function* verdict*, if* VC *holds, then* verdict *provides P with accountability for φ.*

Theorem 15 (Completeness)**.** *For any protocol P, security property φ, and verdict function* verdict*, if* verdict *provides P with accountability for φ, then* VC *holds.*

B. Soundness and completeness of verification conditions

For the proofs see the full version [25, Appendix B].

Lemma 16 (Verifiability)**.**

$$\mathsf{V}_\varphi \iff \mathsf{VE}^{\mathrm{tp}}_\varphi \land \mathsf{VNE}^{\mathrm{tp}}_\varphi$$

Lemma 17 (Minimality)**.**

$$\mathsf{M} \iff \mathsf{M}^{\mathrm{tp}}$$

Lemma 18 (Uniqueness)**.**

$$\mathsf{U} \iff \mathsf{U}^{\mathrm{tp}}$$

Lemma 19 (Sufficiency—Soundness)**.**

$$\mathsf{SF}^{\mathrm{tp}} \land \mathsf{U}^{\mathrm{tp}} \land \mathsf{II} \land \mathsf{RP} \implies \mathsf{SF}$$

Lemma 20 (Sufficiency—Completeness)**.**

$$\mathsf{SF} \land \mathsf{SM} \land \mathsf{V} \implies \mathsf{SF}^{\mathrm{tp}}$$

Lemma 21 (Completeness)**.**

$$\mathsf{VNE}^{\mathrm{tp}}_\varphi \implies \mathsf{C}$$

With the results above, we can proof the central theorems of this work—soundness and completeness of $\mathsf{VC}^{\mathrm{tp}}$.

Theorem 22 (Soundness)**.** *For any protocol P, security property φ, and case tests $\mathcal{C} = \tau_1, \ldots, \tau_n$, if* $\mathsf{VC}^{\mathrm{tp}}$*,* II*, and* RP *hold, then* $\mathrm{verdict}_{\mathcal{C}}$ *provides P with accountability for φ.*

Theorem 23 (Completeness)**.** *For any protocol P, security property φ, and case tests $\mathcal{C} = \tau_1, \ldots, \tau_n$, if* $\mathrm{verdict}_{\mathcal{C}}$ *provides P with accountability for φ, and* SM *holds, then* $\mathsf{VC}^{\mathrm{tp}}$ *holds.*

C. Proof of sufficiency condition for BR

For the proof see the full version [25, Appendix C].

Lemma 24.

$$\mathcal{A} \subseteq PN \land fn(P) \cap PN = \emptyset \implies \mathsf{BR} \tag{14}$$

D. Implications of the Results

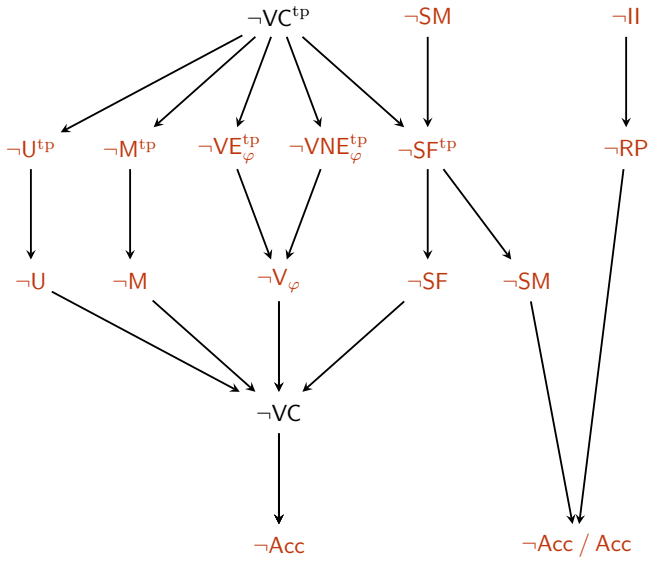
The implications of a failed condition are depicted in Figure 2. Each arrow in the diagram represents an implication; each branch a disjunction. The implications follow from Lemmas 16 to 18 and 20, and Definition 7 as well as the definitions of the respective conditions. For example, if $\neg\mathsf{VNE}^{\mathrm{tp}}_\varphi$, we know by Lemma 16 that $\neg\mathsf{V}_\varphi$. Hence, $\neg\mathsf{VC}$ and by Theorem 15 accountability is not provided.

In the following, we discuss the meaning of each failed condition and give hints on how to fix the problems.

a) Case $\neg\mathsf{SF}^{\mathrm{tp}}_{\tau_i}$: There does not exist a single-matched trace for τ_i in which only a subset of the blamed parties is corrupted. At least one party, which is needed to cause a violation is not blamed. Accountability may still be provided.

Hint: Assume $\mathsf{VNE}^{\mathrm{tp}}_{\varphi,\tau_i}$. If $\neg\mathsf{SM}_{\tau_i}$, we should solve this problem first. In all single-matched traces of τ_i, there exists at least one corrupted party which is not one of the instantiated free variables of τ_i. It may be possible to revise τ_i by adding additional free variables and action constraints such that all parties needed for a violation are blamed by τ_i.

Figure 2. Decision diagram for the requirements and verification conditions defined in Section VII. Each edge represents an implication, each branch a disjunction.



b) Case $\neg\mathsf{VE}^{\mathrm{tp}}_{\varphi,\tau_i}$*:* No case test holds, but the security property is violated. This indicates that the case tests are not exhaustive, that is, capture all possible ways to cause a violation. Accountability is not provided.

Hint: The trace TAMARIN found as a counterexample may give a hint for an additional case test or shows that the security can be violated in an unintended way.

c) Case $\neg\mathsf{VNE}^{\mathrm{tp}}_{\varphi,\tau_i}$*:* The case test τ_i holds but the security property is not violated. This indicates that there exists a trace where τ_i is not sufficient to cause a violation. Accountability is not provided.

Hint: The trace TAMARIN found as a counterexample may give a hint to revise τ_i such that for all traces in which it holds the security property is violated.

d) Case $\neg\mathsf{M}^{\mathrm{tp}}_{\tau_i}$*:* There exists an instantiation of a case test τ_j which blames strictly fewer parties than an instantiation of τ_i in the same trace. Accountability is not provided.

Hint: Assume $\mathsf{VNE}^{\mathrm{tp}}_{\tau_i}$ and $\mathsf{VNE}^{\mathrm{tp}}_{\tau_j}$. If both τ_i and τ_j are necessary for $\mathsf{VE}^{\mathrm{tp}}_{\varphi}$ to hold, they need to be separated such that they do not hold simultaneously. This can be accomplished by replacing τ_i with $\tau_i \wedge \neg\big(\tau_j \wedge [\![fv(\tau_j) \subsetneq fv(\tau_i)]\!]\big)$.

e) Case $\neg\mathsf{U}^{\mathrm{tp}}_{\tau_i}$*:* A party is blamed by an instantiation of τ_i but it has not been corrupted, thereby holding an honest party accountable. Accountability is not provided.

Hint: Assume $\mathsf{VNE}^{\mathrm{tp}}_{\varphi,\tau_i}$. If $\neg\mathsf{M}^{\mathrm{tp}}_{\tau_i}$, we should solve this problem first. The trace TAMARIN found as a counterexample shows which party is blamed without having been corrupted. If the corresponding instantiated free variable can never be corrupted, it can be quantified in τ_i to avoid being blamed. If it can be corrupted for some traces, a closer look on τ_i and the protocol is necessary.

f) Case $\neg\mathsf{SM}^{\mathrm{tp}}_{\tau_i}$*:* There does not exist a single-matched trace for τ_i. Either

(i) there does not exist a trace where τ_i holds, or
(ii) τ_i always holds with multiple instantiations, or
(iii) for all traces there exist another case test which holds at the same time

Accountability may still be provided.

Hint: Assume $\mathsf{VNE}^{\mathrm{tp}}_{\varphi,\tau_i}$. In case (i), τ_i may be ill-defined or contains a logic error. In case (ii), if all the instantiations are permutations of each other, a single-matched trace may be obtained by making τ_i antisymmetric. Then for all instantiations ρ, ρ'

$$t \vDash \tau_i\rho \wedge t \vDash \tau_i\rho' \wedge fv(\tau_i)\rho = fv(ct_i)\rho' \implies \rho = \rho' .$$

If the instantiations are not permutations, at least two disjoint groups of parties are always blamed. This requires a closer look on τ_i and the protocol. In case (iii), it may be possible to merge multiple case tests together for which then a single-matched trace exists.

g) Case $\neg\mathsf{II}^{\mathrm{tp}}_{\tau_i}$*:* The case test τ_i is not injective. There exists an instantiation mapping distinct free variables to the same party. Accountability may still be provided.

Hint: See Example 5 for a way to split τ_i.

We note that for the conditions $\mathsf{SF}^{\mathrm{tp}}_{\tau_i}$, $\mathsf{M}^{\mathrm{tp}}_{\tau_i}$, $\mathsf{U}^{\mathrm{tp}}_{\tau_i}$, and SM_{τ_i}, we assumed above that the case tests satisfy $\mathsf{VNE}^{\mathrm{tp}}_{\varphi,\tau_i}$. If this is not the case, then the case test has a fatal error—it does not always lead to a violation—which renders the other conditions meaningless.