Accountability in the Decentralised-Adversary Setting

Robert Künnemann
CISPA Helmholtz Center for Information Security

Deepak Garg
MPI-SWS

Michael Backes
CISPA Helmholtz Center for Information Security

***Abstract*—A promising paradigm in protocol design is to hold parties accountable for misbehavior, instead of postulating that they are trustworthy. Recent approaches in defining this property, called accountability, characterized malicious behavior as a deviation from the protocol that causes a violation of the desired security property, but did so under the assumption that all deviating parties are controlled by a single, centralized adversary. In this work, we investigate the setting where multiple parties can deviate with or without coordination in a variant of the applied-π calculus.**

We first demonstrate that, under realistic assumptions, it is impossible to determine all misbehaving parties; however, we show that accountability can be relaxed to exclude causal dependencies that arise from the behavior of deviating parties, and not from the protocol as specified. We map out the design space for the relaxation, point out protocol classes separating these notions and define conditions under which we can guarantee fairness and completeness. Most importantly, we discover under which circumstances it is correct to consider accountability in the single-adversary setting, where this property can be verified with off-the-shelf protocol verification tools.

I. Introduction

Trust in other parties is the foundation of all security protocols. In scenarios like electronic voting, certified e-mail, online transactions, or processing of personal data, however, the agents involved cannot be trusted to behave according to the protocol. Nevertheless, if the protocol can detect agents causing security violations, it creates an incentive to avoid malicious deviation from the protocol. The ability of a protocol to provide the necessary information for detection is what we call *accountability*.

In this context, accountability is thus the protocol's ability to point out which parties have been misbehaving. This is different from juridical notions of blame, which also require intent and foreseeability. In Figure 1, we outline how we understand accountability in security protocols and define some terminology. A protocol defines the *normative* behavior of a set of agents $\mathcal{A}$, i.e., how they *should* behave. They may, however, decide to *deviate* from that behavior and thus break some security property φ that we assume to hold otherwise. Should there be a violation of φ, there is a mechanism that defines who (e.g., the public or a trusted party) can identify which parties misbehaved, and how they signal that information. The protocol implements this mechanism. It specifies the normative behavior such that, in effect, the agents are holding each other accountable. The output of this mechanism informs some sort of punishment, which may consist in simply identifying misbehaving parties, excluding them from future protocol runs, in slashing their reputation or in bringing them to court. We concentrate on the correctness of the accountability mechanism implemented by the protocol, not the effectiveness of the punishment, which is outside the protocol verification domain.

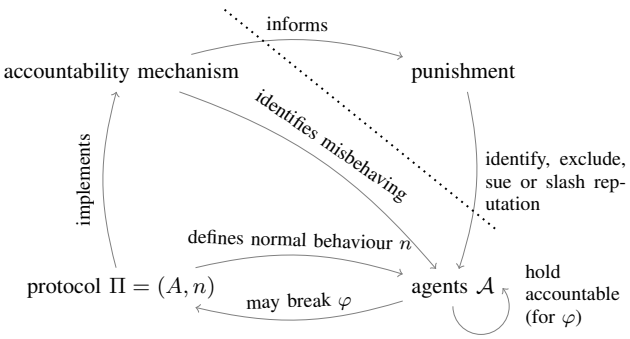

Fig. 1. Approach. The dotted line separates concepts within our model from their use outside of it.

Küsters, Truderung and Vogt [1] pointed out that a major challenge in defining accountability (in the protocol setting) is to distinguish dishonest behavior from misbehavior. Either a party is *honest*, if it follows the protocol, or *dishonest*, if it is controlled by some global adversary and therefore can (but does not have to) deviate from the protocol. A dishonest party is not necessarily *misbehaving* — it may deviate in a completely harmless way. It may even behave like an honest party.

Künnemann, Esiyok and Backes formalize the distinction between actual misbehavior and the mere ability to misbehave, reaching a definition of accountability in the centralized-adversary setting [2]. In this setting, a single adversary is controlling all dishonest parties. In this work, we consider the decentralized setting, in which individual protocol parties can choose to deviate from the protocol. Like Künnemann et al., we say that a party misbehaves if the fact that it deviates causes a violation of some given security property.

To summarize: a priori, parties are assumed either honest or dishonest. At runtime, dishonest parties can deviate, i.e., not follow the protocol. If, doing that, they cause a violation, they misbehave.

To allow individual parties to deviate, we modify the applied-π calculus [3]: subprocesses are annotated with the

effectuating parties and, before the protocol starts, any subprocess that belongs to a deviating party can be replaced by another process. We also examine weaker versions of accountability that consider only deviations that implicate minimal subsets of parties, or where deviating parties share as little information as they can. We relate these notions to each other with separating examples and equivalence proofs. Exploring these notions, we discover:

1) Accountability, in general, is impossible to achieve due to the ability of parties to make themselves dependent on each other.
2) There is, however, a class of optimality notions that leads to applicable definitions and guarantees soundness, i.e., any party in the verdict did deviate. Some of these notions even guarantee a weak form of completeness, i.e., certain deviating parties are guaranteed to appear in the verdict.
3) Optimality w.r.t. simple deviations, where deviating parties cannot have conditionals and are not communicating with each other is equivalent to accountability in the centralized-adversary setting.

The third result establishes the correctness of Künnemann et al.'s verification methodology [2] in the centralized-adversary setting. Their method allows the automated verification of practical protocols like OCSP stapling and Certificate Transparency. Most protocol verification tools model the adversary as a non-deterministic reduction step for message input, with the side condition that the message has to be 'deducible,' i.e., can be derived from previous protocol output with a set of rules. To model party deviation, they transform the process so parties can expose their secrets on the public channel, but are registered as 'corrupted.' By showing that this transformation, in the centralized-adversary setting, is equivalent to accountability in the decentralized setting, we show that accountability w.r.t. simple deviations applies to real protocols.

Moreover, we give an interpretation to these existing results: the centralized setting, in fact, ignores important characteristics of the real world, where colluding parties have to communicate to share secrets and can make their behavior dependent on the behavior of others. The equivalent notion in the decentralized setting — accountability w.r.t. simple, knowledge-optimal deviations — precisely defines what is lost in the centralized-adversary setting.

Taking the existing verification results into account, the first result — impossibility of accountability in the decentralized-adversary setting (without optimality assumptions) — shows that the widely held belief that a single, corrupting adversary is an adequate model for dishonest participants may not always be correct. There is a trade-off between efficient verification and a faithful modeling of misbehavior. In the following, we precisely define what we give up for automated verification.

II. Related work

Accountability describes related but slightly different concepts, depending on the context. Papanikolaou and Pearson [4] investigate the use of the term in numerous scientific fields and applications. Depending on the context, they conclude, accountability is defined using the elements of disclosure, liability and non-repudiation, although, in some contexts, these are only seen as measures to implement accountability. There is an agreement on the overarching goal, however: entities such as organizations or individuals need to make their actions transparent and thus be rewardable or punishable for them. In this work, we concentrate on the mechanism informing this punishment and its correctness (see Figure 1 for an illustration). This is in contrast to, e.g., Feigenbaum et al. [5], who characterize accountability in terms of the effect of some punishment. They point out that a punishment can preserve the anonymity of parties, while a mechanism that identifies parties cannot. We agree with this point, but add to it that the punishment mechanism needs to be informed whom to punish. We thus focus on the correctness of this information. The punishment can consist of the identification of offenders, but also their exclusion from future protocol runs or the slashing of reputation within the protocol. Other forms of punishment can manifest themselves outside of the protocol, e.g., legal procedures. Our model is neither suitable for the modeling of such punishments, nor a quantitative analysis of the punishment's effect.

This is in line with much of the work on accountability in the protocol context, most of which provides or uses a notion of accountability in either an informal way, or tailored to the protocol and its security [6]–[9]. Capturing completeness, i.e., the property that all misbehaving parties are blamed, is particularly challenging.

Defining completeness. Some prior work treats all deviations as misbehavior. This is unproblematic when considering soundness [1]: whenever the judge gives some verdict, every party mentioned has deviated from the protocol. The approach fails, however, when expressing completeness: every party that deviated ought to be in the verdict. Parties can deviate in some way that the judge cannot recognize, especially in the protocol security setting. Küsters et al. identify and tackle this problem (see below), while other approaches still follow this idea, considering any trace that the honest behavior cannot produce to be malicious behavior [10], [11].

Treating all (visible) deviations as misbehavior works fine for distributed systems, where the goal is masking *faults*. Systems like PeerReview [10] can guarantee completeness in the Byzantine setting, by monitoring all participating systems. Security protocols are also analyzed in the Byzantine setting; it is, however, hopelessly unrealistic to assume all communication can be monitored. There is simply no way that a web server can be sure that two clients are not communicating with each other outside the protocol. Even if all communication was public — which is unrealistic — any accountable protocol would need to verify the behavior of every party in full detail, even if this party is not involved in later phases or cannot meaningfully disrupt communication. A voter in an e-voting protocol would have to refrain from any communication between casting their votes and the final tally. Systems like PeerReview can therefore only apply to faults

in distributed systems [10], where the component's adherence to specification is a design goal. Jagadeesan et al. admit that in their model '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' [11]. Küsters et al. [1], by contrast, stress the difference between dishonest agents and misbehaving agents and propose a completeness definition that, like ours, is centered around the security property. They propose a middle ground between a protocol-specific definition and a fully protocol-agnostic definition. Their definition is parametric in a use-case-specific policy that indicates who ought to be blamed in which case. These policies, however, cannot reliably express completeness in a protocol-agnostic way. In case of joint misbehavior, the policy is either incomplete, unfair, or it encodes the accountability mechanism itself (see Appendix B).

Accountability from causation between protocol events Instead of equating misbehavior and corruption, we consider misbehaving parties to be those whose deviation (as a whole) causes a violation. By contrast, other causality-based approaches consider individual protocol actions as causes for security violations [5], [12], [13]. With this method, unrelated events can be removed prior to forensic analysis. It was considered [13] to be a promising first step for defining accountability: once the cause trace, i.e., the sequence of protocol events that are sufficient to cause a violation, is computed, we hold all involved parties accountable. But not all protocol actions that are causally related to an attack are malicious. A key server distributing public keys can be necessary in the attack because the attack relies on public keys, without even deviating from the protocol. While cause traces may help in filtering out parties that were altogether uninvolved in the attack, they nonetheless refer us to the original question: What constitutes malicious misbehavior? Datta et al. [13] sketch a procedure for blame assignment where the key server is blamed unless he followed the protocol to the letter. This is overly strict (a deviation from the protocol might be unrelated to the attack) and again requires complete monitoring. Feigenbaum et al. [5] consider causation on protocol events a black box and focus on punishment (possibly protecting the parties' identities). Gössler et al. [12] focus on traces and assume an explicit dependency relation to recognize faulty components in a system, therefore excluding the case of parties colluding privately. Their approach hence applies to distributed systems, but not the security setting.

Accountability in the centralized setting Künnemann et al. [2] provide a verification methodology for accountability in the centralized-adversary setting, which is what all widely used protocol verification tools are based on. They used this methodology to verify practical protocols (OCSP stapling and Certificate Transparency) and academic proposals. The present work provides a foundation and justification for their methodology, as the centralized-adversary setting can neither reflect individual deviations, nor communication between deviating parties. We formally relate their ad hoc modeling of protocol deviations via party corruption to the much richer setting where parties are processes that communicate with each other. More precisely, we show equivalence to a weakened version of accountability. Hence their verification results transfer to accountability in the decentralized setting, the definition we present here. We will also show that (strong) accountability in the decentralized setting is impossible to achieve without restrictions, further emphasizing the loss of generality in the centralized-adversary setting.

Summarizing, existing approaches are either protocol specific, or require the slightest deviation to be punished, even if the normative behavior had had the same causal dependencies. In scenarios where parties can communicate privately, however, the latter approach reaches its limits. Analysis methods exist in the centralized-adversary setting [2], but as this setting is not accurately reflecting the actual threat — multiple, coordinating protocol participants — it is unclear what they compute.

III. Process calculus

To reason about accountability formally, we must reason about deviating parties. In the following, we introduce a process calculus that allows individual parties to deviate from the protocol. The calculus draws heavily from the applied-π calculus [14], but it annotates processes with the party executing them. We describe a protocol in terms of the structure of the process, which determines which parties run in parallel and which parties share secrets, and the *normative behavior*, which determines which process each party runs. Any party may choose to run a different process, which we call a *deviation*. We will first introduce the term algebra and the process calculus (closely following the applied-π calculus) before we specify how parties deviate.

a) Notational conventions: We denote domain restriction of a function f to a subset $S \subseteq dom(f)$ as $f|_S$. We filter a sequence l by a set S, denoted $l|_S$, by removing each element that is not in S.

b) Terms and equational theories: We model messages by abstract terms. Assume a countably infinite set of names $\mathcal{X}$ used to model cryptographic keys and nonces, and a countably infinite set of variables $\mathcal{V}$. Given a signature Σ, i.e., a set of function symbols with an arity each, we write f/n for function symbol f of arity n. Let Terms_Σ be the set of terms built over Σ, $\mathcal{X}$, and $\mathcal{V}$, and $names(t)$, respectively $vars(t)$, denote the set of names, respectively variables, appearing in a term t. We denote the set of ground terms, i.e., terms without variables, by $\mathcal{M}_\Sigma$. When Σ is fixed or clear from the context, we omit it and simply write Terms for Terms_Σ and $\mathcal{M}$ for $\mathcal{M}_\Sigma$.

We equip the term algebra with an equational theory E, i.e., a finite set of equations of the form $m = m'$ where $m, m' \in$ Terms. The equational theory defines the binary relation $=_E$ on terms, which is the smallest equivalence relation containing the equations in E that is closed under application of function symbols, bijective renaming of names and substitution of variables by terms.

Example 1. The signature $\Sigma_{\text{sig}} := \{true/0, pk/1, sig/2, versig/3, extract/1\}$ and the following equational theory model digital signatures: $versig(pk(x), sig(x, y), y) = true$ and $extract(sig(x, y)) = y$.

Let E and Σ be fixed such that they include function symbols for construction and deconstruction of pairs, i.e., $\Sigma \supseteq \Sigma_{\text{pairs}} := \{\langle .,. \rangle, \pi_1, \pi_2\} \subseteq \Sigma$ and equations $\pi_1(\langle x, y\rangle) = x$ and $\pi_2(\langle x, y\rangle) = y$ are in E. We use $\langle x_1, x_2, \ldots, x_n\rangle$ as a shorthand for $\langle x_1, \langle x_2, \langle \ldots, \langle x_{n-1}, x_n\rangle \ldots\rangle$. Set membership modulo E is denoted by $\in_E$ and defined as $e \in_E S$ iff $\exists e' \in S.\ e' =_E e$. A substitution σ is a partial function from variables x_i to terms t_i written $\sigma = \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$. We homomorphically extend σ to apply to terms and use postfix notation to denote its application, e.g., $t\sigma$ applies σ to t.

c) Grammar and operational semantics: In contrast to the applied-π calculus [14], our calculus incorporates the effectuating party of a process. Since processes running in parallel can represent threads or programs running in parallel as well as computers in an asynchronous network, this annotation is necessary. Assume a set of parties $\mathcal{A}$, a subset of which, $\mathcal{T} \subset \mathcal{A}$, is trusted never to deviate from normative behavior.

$\langle P,Q\rangle$ ::= (plain processes)	$\langle A,B\rangle$ ::= (ext. process)
0	$A \mid B$
$\nu n; P$	$\nu n; A$
$\text{in}(x);\ P$	$\nu x; A$
$\text{out}(m);\ P$	$\{^m/_x\}$
if $m = m'$ then P else Q	$\cdot_p$ for $p \in \mathcal{A}$
event $t;\ P$	P

Fig. 2. Syntax, where $n \in \mathcal{X}$, $x \in \mathcal{V}$ and $m, m' \in$ Terms

Plain processes (defined by the grammar in Figure 2) usually define the behavior of a single party as a combination of message input and output (on a single, global, public channel), conditionals w.r.t. $=_E$ and scope restriction $\nu n; P$ (which creates a fresh name n and then behaves like P). Furthermore, plain processes can emit events, which can be used to model signaling behavior (see Example 2 below) or append-only logs. We will omit else branches with zero processes for brevity. For simplicity, we exclude parallel composition and replication in plain processes, as otherwise we would need to track several processes per party[1]. Extended processes combine extended or plain processes via parallel composition ($A \mid B$). We use scope restrictions on names to distribute shared secrets. An active substitution $\{^m/_x\}$ acts as a 'floating' substitution operation. Together with scope restriction on variables, they allow processes to transmit terms between each other (see below). As usual, names and variables have scopes delimited by restrictions and inputs. We write $fv(A)$, $fn(A)$ for the set of free variables and names of A, respectively. Finally, the 'hole' $\cdot_p$ serves as a placeholder for a subprocess. Later, we define how the normative behavior of a party $p \in \mathcal{A}$ or a deviation may substitute this hole by a process. We thus require the following condition.

[1]This could be achieved, e.g., by drawing fresh names as session identifiers [15].

Definition 1 (skeleton process)**.** A *skeleton process* is defined by the grammar for extended processes without the last production rule (which includes plain processes) and exactly one hole $\cdot_p$ per party $p \in \mathcal{A}$.

A protocol is now defined in terms of a skeleton process determining how information, i.e., names and terms, are initially shared between parties, and a function that maps every party to a plain process.

Definition 2 (protocol)**.** A protocol $\Pi = (A, n)$ consists of a skeleton process A and a function n from $\mathcal{A}$ to plain processes such that for all $p \in \mathcal{A}$, all $fn(n(p))$ and $fv(n(p))$ are bound in the scope of $\cdot_p$ in A. We call $n(p)$ the *normative behavior* of p.

A straightforward approach to achieve accountability is to have a trusted monitor, which executes requests but expects them to be signed. The next example follows this paradigm.

Example 2 (delegation example)**.** Assume the signature Σ consisting of Σ_{pairs}, Σ_{sig}, $\Sigma_{\text{log}} = \{Log/2, Exec/1\}$ and $\Sigma_{\text{act}} = \{NAct/0, SAct/0, UnAct/0, isAct/1\}$. Consider the equations in Example 1 and the following: $isAct(a) = true$ if a is either a normal action ($NAct$), special action ($SAct$) or an unusual action ($UnAct$). Assume four parties, A, B, I and T. Among those only $\mathcal{T} = \{T\}$ is trusted. The following skeleton process defines the generation of A and B's signing keys and the distribution of their public parts:

$$\nu sk_A; \nu sk_B; \{^{pk(sk_A)}/_{pk_A}, ^{pk(sk_B)}/_{pk_B}\} \mid \cdot_A \mid \cdot_B \mid \cdot_I \mid \cdot_T$$

The party A processes two kinds of actions, normal actions and special actions. Normal action are signed and sent to T for execution. Special actions are forwarded to B for authorization, which signs the request identifier n_a. Finally, A sends both signatures to T for processing.

$$\begin{aligned} n(A){:=}\ &\text{in}(a);\ \text{if } a = NAct \text{ then } \text{out}(\langle a, sig(sk_A, a)\rangle) \\ &\text{else if } a = SAct \text{ then} \\ &\quad \nu\ n_a;\ \text{out}(\langle n_a, sig(sk_A, \langle a, n_a\rangle)\rangle); \\ &\quad \text{in}(r);\ \text{if } versig(pk_B, r, n_a) = true \text{ then} \\ &\qquad \text{out}(\langle a, n_a, sig(sk_A, \langle a, n_a\rangle), r\rangle) \\ n(B){:=}\ &\text{in}(m);\ \text{if } versig(pk_A, \pi_2(m), \langle SAct, \pi_1(m)\rangle) = true \\ &\quad \text{then } \text{out}(sig(sk_B, \pi_1(m))) \end{aligned}$$

The trusted monitor T verifies the signature of incoming requests, but not which action they contain. But it takes note of who issued the request. This strategy is typical in scenarios where security violations may only be determined after the fact, e.g., if T cannot tell usual from unusual actions. For example, in a hospital, it may be unusual for a doctor (A) to retrieve data belonging to another doctor's (B's) patient. Still, in case of an emergency, A should be able to request this special action without further ado, but has to justify it later. The responsibility of T is to enforce that the necessary information is present. Let xa, xn, s_1, s_2 in the second branch abbreviate $xa = \pi_1(m)$, $xn = \pi_1(\pi_2(m))$, $s_1 = \pi_1(\pi_2(\pi_2(m)))$ etc. such that $m = \langle xa, xn, s_1, s_2\rangle$. Then,

$$\textsc{Then}\quad (\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q)_{p_A} \xrightarrow{p_A} P_{p_A} \quad \text{if } t_1 =_E t_2$$

$$\textsc{Else}\quad (\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q)_{p_A} \xrightarrow{p_A} Q_{p_A} \quad \text{if } t_1 \neq_E t_2$$

$$\textsc{Comm}\quad (\mathsf{out}(x); P)_{p_A} \mid (\mathsf{in}(x); Q)_{p_B} \xrightarrow{(p_A, p_B, x)} P_{p_A} \mid Q_{p_B}$$

$$\textsc{Event}\quad (\mathsf{event}\ x; P)_{p_A} \xrightarrow{(p_A, x)} P_{p_A}$$

Fig. 3. Reduction rules

$n(T)$:=$\mathsf{in}(m)$;
if $versig(pk_A, \pi_2(m), \pi_1(m)) = true$ then
event $Log(A, \pi_1(m))$; event $Exec(\pi_1(m))$
else if $versig(pk_A, s_1, \langle xa, xn\rangle) = true$ then
if $versig(pk_B, s_2, xn) = true$ then
event $Log(\langle A, B\rangle, xa)$; event $Exec(xa)$

The party I with $n(I) := \mathsf{out}\ (SAct);\ \mathsf{out}\ (\langle pk_A, pk_B\rangle)$ models an intruder who knows A's and B's public key and triggers communication between A and B leading to a normal run. □

The operational semantics is same as for the applied-π calculus, except that *a*) the topmost process inserted at a hole $\cdot_p$ is annotated with the party p (skipping scope restrictions, i.e., $\nu n.P$ becomes $\nu n.(P)_p$), and reduction preserves these annotations. *b*) Reductions are additionally labeled with the effectuating party for internal reductions, and with the sending party and the recipient in case of communication (see Figure 3). Appendix A recalls the applied-pi calculus, including our modifications, in detail. Let $traces(A) = \{(l_1, \ldots, l_n) \in ((\mathcal{A} \times \mathcal{A} \times \text{Terms}) \uplus (\mathcal{A} \times \text{Terms}) \uplus \mathcal{A})^* \mid A \xrightarrow{l_1} \cdots \xrightarrow{l_n}\}$. We define

$$ctl(e) = \begin{cases} (p_A, p_B) & \text{if } e = (p_A, p_B, m) \\ (p_A) & \text{if } e = (p_A, m) \\ (p_A) & \text{if } e = p_A \end{cases}$$

and lift it to traces to define the control flow $ctl(t) \in ((\mathcal{A} \times \mathcal{A}) \uplus \mathcal{A})^*$ of a trace. Unless stated otherwise, we will assume the set of visible event (visible to some judge) to be $V = \mathcal{A} \times \text{Terms}$. We will impose that verdicts are derived from the visible part of a trace $t|_V$. The control flow, $ctl(t)$, will be used to define 'what-if' scenarios that are related to the actual trace.

d) Deviations: A deviation is a function that overwrites the normative behavior of untrusted parties and may only refer to names and variables the normative behavior has access to. For instance, a deviation for A in Example 2 may refer to $pk_B \in fv(n(A))$, which is bound to $pk(sk_B)$, but not to $sk_B \notin fn(n(A))$.

Definition 3 (protocol deviation / instance)**.** For a protocol $\Pi = (A, n)$, a partial function δ from $\mathcal{A} \setminus \mathcal{T}$ to plain processes such that $fv(\delta(p)) \subseteq fv(n(p))$ and $fn(\delta(p)) \subseteq fn(n(p))$ for every $p \in dom(d)$ is called a *deviation* and induces an instance of Π, which we denote $\Pi\delta$. In $\Pi\delta$, each occurrence of $\cdot_p$ is substituted by $\delta(p)$, if defined, and $n(p)$ otherwise. In both cases, the first subprocess that is not of the form $\nu m; P$ is annotated with p.

For the empty deviation $\emptyset$, the instance $\Pi\emptyset$ is an extended process that contains only the normative behavior of all parties. By Def. 2, the free names and variables of all plain processes in the domain of n are bound in the scope of $\cdot_p$ in A, hence any instance of the protocol is closed.

Given a protocol Π and deviation δ, we consider security properties as predicates over traces $t \in traces(P\delta)$. For all security properties φ, we require that φ is congruent w.r.t. E and only regards the visible part of the trace, i.e. for all t, t', $t|_V =_E t'|_V \implies \varphi(t) = \varphi(t')$.

Example 3. The delegation protocol (Ex. 2) shall hold parties accountable for effectuating actions that are neither normal nor special. The security property is thus:

$$\varphi(t) := \forall e.\ (T, e) \in t|_V \wedge e =_E Exec(a) \implies a \in_E \{NAct, SAct\}.$$

IV. Accountability

With this calculus in place, we can now argue about accountability in traces where one or more parties decide not to follow the protocol. By restricting the domain of a deviation δ, we can even examine *counterfactual* scenarios where only some of the deviating parties are exhibiting the same behavior, while the other parties are reset to their normative behavior. We will use this feature to define the *a posteriori verdict* (short: apv). The apv specifies which parties *should* be held accountable, but can only be computed if the behavior of each party is known. It forms the basis for our later treatment of unknown deviations: Ideally, an accountability mechanism should always give a verdict that coincides with this a posteriori judgment. An accountability protocol is thus a protocol that computes the apv.

A. A posteriori verdict

We define accountability as the ability of a protocol to compute which parties caused a security violation. Accountability is hence a meta property; we speak of accountability for some security property φ.

To decide which parties caused the violation, we follow the structured-model approach to causation [16]. The idea is to determine causal relations by intervening on potential causal factors. In our case, the only causal factor we intervene on is whether a party deviates at all.

Most definitions of actual causation follow the counterfactual approach [16], [17] going back to Lewis [18] and possibly Hume [19, Section VII]. These definitions all follow the idea that 'event A causes event B' means that B would not have happened had it not been for A. In the structured-model approach, the factual scenario is modified (*intervened on*) to remove A. If B never occurs after this so-called intervention, A was necessary to cause B.

We build our definition on the dual notion of sufficient causation [13], [20]. It captures all parties for which the *fact that they are deviating at all* is causing the violation, i.e., events of the form 'a party or a set of parties S $\subseteq \mathcal{A}$ are

deviating' are sufficient to cause $\neg\varphi$. Intuitively, such an event is a cause for a violation iff:

SC1. A violation indeed occurred and S indeed deviated.

SC2. If all deviating parties, *except the parties in* S, behaved honestly, the same violation would still occur.

SC3. S is minimal, i.e., SC1 and SC2 hold for no strict subset of S.

SC1 is a prerequisite for any form of causation. SC2 formalizes that the deviation of the parties in S alone is *sufficient* to disrupt the protocol. SC3 removes parties who deviated, but whose deviations are not needed to produce a violation. Consider, e.g., two parties, A and B, colluding against a secret sharing scheme with a threshold of two. Removing either A or B from $\mathrm{S} = \{A, B\}$ would eliminate the violation. Hence A and B jointly caused the violation. Now consider two parties, C and D, running two attacks on a different system, at different points in time, without coordination. Due to SC3, $\{C, D\}$ would not be considered a cause, but both $\{C\}$ and $\{D\}$. Hence C and D *independently* caused the violation. Distinguishing between joint and independent causation is a strength of sufficient causation. We define the a posteriori verdict (apv) as the set of all sets of parties that caused a violation.

Definition 4 (a posteriori verdict)**.** Given a protocol $\Pi = (A, n)$, a property φ, and a deviation δ, the *a posteriori verdict* for $t \in \mathit{traces}(\Pi\delta)$ is defined $\mathit{apv}_{\Pi,\varphi}(t, \delta) :=$

$$\{\mathrm{S} \mid \neg\varphi(t) \text{ and } \mathrm{S} \subseteq \mathit{dom}(\delta) \quad \text{(SC1)}$$
$$\text{and S is minimal s.t.} \quad \text{(SC3)}$$
$$\exists t' \in \mathit{traces}(\Pi\delta|_{\mathrm{S}}).t =_{\mathrm{S}} t' \wedge \neg\varphi(t')\}. \quad \text{(SC2)}$$

Here, $t =_{\mathrm{S}} t'$ is short for $\mathit{ctl}(t)|_{\rightleftharpoons \mathrm{S}} = \mathit{ctl}(t')|_{\rightleftharpoons \mathrm{S}}$, where $\rightleftharpoons \mathrm{S} := (\mathrm{S} \times \mathcal{A}) \cup (\mathcal{A} \times \mathrm{S}) \cup (\mathrm{S})$ denotes the part of the control flow of t that involves any party in S.

The intervention on the trace in SC2 is characterized by restricting the domain of δ to S. We hence consider a run of the protocol where some parties — those outside of S — are reverted to their normative behavior. If the way S deviated is sufficient to cause a violation, such a run must exist. But note that this run is not arbitrary; the parties in S are bound to their previous control flow. By contrast, parties that are reverted to their normative behavior can have any control flow, as the process defining their deviating behavior may have an entirely different structure from their normative behavior.

Example 4. Consider a deviation δ for Example 2 with $\delta(B) := \mathsf{out}(sk_B)$ and $\delta(A)$ defined as follows:

$\mathsf{in}(xsk);\ \nu n_a;$
$\mathsf{out}(\langle \mathit{UnAct}, n_a, \mathit{sig}(sk_A, \langle \mathit{UnAct}, n_a\rangle), \mathit{sig}(xsk, n_a)\rangle)$

Let trace t be such that B sends her signing key to A, who fakes B's authorization and then instructs T to execute an unusual action. W.r.t. the property that no unusual action was executed, i.e., $\varphi(t) := (T, \mathit{Exec}(\mathit{UnAct})) \notin_E t$, the a posteriori verdict $\mathit{apv}(t, \delta)$ is $\{\{A, B\}\}$, as reverting either A or B to their normative behavior avoids φ. By contrast, if $\delta'(A) := \mathsf{out}(\mathit{sig}(sk_A, \mathit{UnAct}))$ and B shares its signing key with I, i.e., $\delta'(B) := \mathsf{out}(sk_B)$ and $\delta'(I) := \mathsf{in}(m)$, then $\mathit{apv}(t', \delta') = \{\{A\}\}$, for the obvious trace t, as B's behavior, even if it was reckless, had no bearing on the coming about of $\neg\varphi$.

B. Discussion

Causation versus (observable) deviation. Some causation-based definitions in distributed systems consider any deviation from specification a fault [10]. This concept does not translate to misbehavior in the security setting, where the behavior of protocol participants is not fully observable. A server, e.g., only observes the messages that clients send to her, but not messages between clients. Causation weakens this requirement: hidden messages are irrelevant unless they are causally relevant to the attack.

Notion of causation. While no universally accepted notion of causation exists, Lewis's counterfactual approach [18] is the inspiration for most definitions, e.g. [16], [17], [21]. All of these can be abstracted to a form of 'necessary causation' that is dual to sufficient causation [13], [20] as presented here: sets of necessary causes can be mapped to sets of sufficient causes and vice versa. In particular Halpern and Pearl's definition [16], [17], the most popular to date, can be interpreted in terms of sufficient causation where interventions in SC2 may not stray from the original control flow [22].

Valid counterfactuals. The result of the intervention in SC2, also called *counterfactual*, has to exhibit the same control flow as the actual trace. In the context of programs and distributed systems, this is fairly common [13], [22], [23], but other methods exist, e.g., based on their output according to specification [24], [25]. This approach has little use in the wider context of philosophy and law studies (what would be the 'control flow' of a car accident?) Various notions of causality exist whose main difference is in which counterfactuals are permitted. They can be based on the structure of the causal graph [16], [17], or on a fine-grained classification of events [26], [27].

Hyperproperties. Many important security properties, e.g., information flow, can only be expressed as hyperproperties [28]. The apv builds on sufficient causation. Like its dual [16], it deals with causation between events defined on single traces. Investigations in the cryptographic settings [29] show that a transfer of the apv is not straightforward, as it is unclear how to define the counterfactual w.r.t. a set of traces that violate the hyperproperty. Indeed, the definition of causality in the probabilistic setting is the subject of ongoing research [30]. First results for causal analysis of hyperproperties have only just been made [31].

C. Verdict function

We expect security protocols Π to be equipped with a function $\mathit{verdict} : \mathit{traces}(\Pi) \to 2^{2^{\mathcal{A}}}$ that assigns each trace a verdict. Like security properties, we define verdicts to be congruent modulo E and to consider only events. In practice, the use case can impose additional requirements: e.g., for a

verdict to be computable from a public log, or from events output by a trusted party, etc. The verdict function abstracts whatever entity is giving the verdict, be it a real judge or jury, or a designated party which is part of the protocol. To clarify its semantics, we sometimes write a verdict v as a propositional formula with parties as atoms: $\bigvee_{C \in v} \left(\bigwedge_{p \in C} p \right)$.

The verdict provides the set of *independently* accountable groups of agents such that all agents within such a group are *jointly* accountable. For example, a verdict $(A \wedge B) \vee (B \wedge C) \vee (A \wedge C)$ (equivalently, $\{\{A, B\}, \{B, C\}, \{A, C\}\}$) states that the way any *two* of the three parties A, B, C actually deviated was sufficient to provoke the violation. This occurs, e.g., when a simple majority was enough to accept a faulty input, and all three did in fact misbehave.

Example 5 (verdict)**.** In Example 2, the task of the monitor is to supply sufficient evidence of the parties deviating from protocol by issuing an unusual action. We thus assume only the events of the trusted party to be visible, $V = \{T\} \times \text{Terms}$ and, using the shorthand $e \in^T t \iff (T, e) \in t|_V$, we consider the verdict function $\mathit{verdict}(t) :=$

$$\begin{cases} \{\{A, B\}\} & \text{if } \mathit{Exec}(a), \mathit{Log}(\langle A, B\rangle, a) \in^T t \\ & \quad \wedge\, a \notin \{\mathit{SAct}, \mathit{NAct}\}, \\ \{\{A\}\} & \mathit{Exec}(a), \mathit{Log}(A, a) \in^T t \wedge a \neq \mathit{NAct}, \\ \emptyset & \text{otherwise.} \end{cases}$$

D. Definition of accountability

A protocol provides accountability for a property if there is a verdict function for this property, and this verdict function is sound and complete.

Definition 5 (accountability)**.** We say that a function $\mathit{verdict} : \mathit{traces}(\Pi)|_V \to 2^{2^A}$ provides a protocol $\Pi = (A, n)$ with accountability for a property φ if for any deviation δ and $t \in \mathit{traces}(\Pi\delta)$,

$$\mathit{verdict}(t) = \mathit{apv}(\Pi, d).$$

The verdict function should be correct for any visible trace produced by the protocol, but many deviations produce the same trace. Nevertheless, a truly accountable protocol ought to provide a verdict with certainty. These are conflicting goals, as we will find in the next section, and later resolve by considering only optimal deviations. The present definition reflects the certainty requirement because $\mathit{verdict}$ is a function on traces. Hence all deviations reproducing the trace need to have the same a posteriori verdict.

Example 6 (Whodunit)**.** S and A coordinate on some fixed name $a \in \mathcal{X}$ as follows. $n(S)$ sends a to A; $n(A)$ receives this value. Both report it to a trusted party T; only messages sent to T are visible. In the actual trace, $\delta(A)$ sends an a^* different from S's reported choice to T. In this case, a correct verdict is impossible without additional information on S's and A's deviation, as two minimal scenarios are plausible: S deviated and sent a^* to A, but reported a, or A deviated and received a, but reported a^* to blame S. Both deviations entail a different apv, but produce the same trace. Thus, no verdict function can provide accountability.

We stress the difference between uncertainty and disjunctions in the verdict. If the verdict is $\{\{S\}, \{A\}\}$, i.e., $S \vee A$, it means that both a deviation of S or a deviation of A on their own are sufficient to cause $\neg\varphi$, but the trace indicates that both S and A deviated.

E. Discussion

Relation to verifiability. Küsters et al. pointed out the relationship between verifiability and accountability by relating the verdict to whether a violation occurred [1].

Corollary 1 (accountability implies verifiability)**.** Def. 5 and 4 imply: If $\mathit{verdict}$ provides Π with accountability for some φ, then it also provides verifiability, i.e., for any deviation δ and $t \in \mathit{traces}(\Pi\delta)$, $\varphi(t) \iff \mathit{verdict}(t) = \emptyset$.

Trust assumptions. Trust in other parties or secure computing environments (e.g., SGX enclaves) is reflected in the set of trusted parties $\mathcal{T} \subseteq \mathcal{A}$. Two protocols may be compared in terms of the parties they trust if they both define the same skeleton process.[2]

Probabilistic accountability. While the apv cannot capture causes for (violations of) hyperproperties, accountability could, in principle, be limited to defined subsets of $\mathit{traces}(\Pi)$. That way, we could capture probabilistic accountability if the underlying calculus were probabilistic.

Network attackers. The absence of private channels in our calculus restricts our definition to situations where all parties have equal access to the network. Communication is nondeterministic, so any party can receive any message and therefore all dishonest parties can act as network attackers. To consider a 'pure' network attacker, we may define a dishonest party N with $n(N) := 0$. As $\mathit{fv}(n(N)) = \mathit{fn}(n(N)) = \emptyset$, any attack mounted by N can be mounted by any other dishonest party. A protocol that permits attacks by N therefore cannot provide accountability (or any relaxation we discuss in the follow-up) unless N is the only untrusted party. In that case, accountability becomes equivalent to verifiability (see Corollary 1), which is easier to verify directly.

Intrinsic limitations. Besides the aforementioned remark about communication and obvious constraints due to the choice of the symbolic setting, the use of the apv imposes a restriction on trace properties (see Discussion in Section IV-B). A computational variant of this definition is conceivable, but technically challenging.[3] Furthermore, one of the relaxations in Section IX imposes conditional-free processes. They are delicate to define when Turing machines constitute the underlying model of computation.

[2]Protocols (Def. 2) need to include $\mathcal{T}$, which was omitted for simplicity.

[3]Morio and Riahi managed to transform Künnemann et al.'s definition against centralized attackers into a cryptographic game [29], [32] but employed Künnemann et al.'s verification conditions as a trace-based characterization of accountability. Whether such a characterization of Def. 5 exists is an open question.

F. The trouble with provocation

Unfortunately, accountability is impossible to achieve if deviating parties can communicate privately. In the following example, B sends a message to A to provoke it to mount an attack.

Example 7. Consider a deviation δ with $dom(\delta) = \{A, B\}$ and a function symbol $go/0$:

$\delta(A) =$in(m); if $m = go$ then event $Boom()$ else *behave honestly*
$\delta(B) =$out(go)

and some security property that A can break on its own, e.g., $\varphi(t) \iff (A, Boom()) \notin t$. For a trace t where a violation occurs, we have $apv(\delta, t) = \{\{A, B\}\}$. If *verdict* cannot take the communication between A and B into account — e.g., if it is defined on events $V = \mathcal{A} \times$ Terms — then it cannot distinguish δ and a δ' with $dom(\delta') = \{A\}$ and

$\delta'(A) :=$ event $Boom()$

Informally speaking, A can plausibly discredit B, saying that it was 'provoked' to attack.

This problem arises whenever private communication (w.r.t. V) with any dishonest party is possible. Hence, unless one were willing to assume all messages were public, no nontrivial protocol can provide accountability. Note that this example is not an artifact of our definition. In δ, it is intuitive that A and B are causing the violation in δ, both w.r.t. the actions they perform and their decision to deviate at all. If A could prove that it ran the process $\delta(A)$, it would be plausible to hold B accountable, too.

In the centralized-adversary setting, it is possible to achieve accountability [2]. The provocation example is simply not expressible in this setting, because the network adversary is modeled in terms of a deduction relation. Loosely speaking, the adversary is merely a sequence of messages deducible from the protocol's output up to that point. This adversary cannot make its actions causally depend on each other. The *possibility* of accountability is therefore an artifact of how the adversary is represented. In Section IX, we will see that the centralized setting is equivalent to one of the optimality notions we will discuss in the next section.

V. Accountability under optimality assumptions

In this section, we investigate how to avoid the provocation example by assuming the deviating parties behave optimally. We will discuss three notions of optimality: verdict optimality, which considers only deviations with minimal apvs, knowledge optimality, which considers only deviations with minimal communication between deviating parties, and simple deviations, where processes cannot branch.

The idea is that an imaginary 'judge', who cannot know the processes that A and B ran in Example 7, has no evidence that points to B. This judge would thus look for the simplest explanation to the observations she made.

What we deem simple should be convincing to the public; it becomes part of the accountability definition. All three approaches address Example 7: δ is neither optimal w.r.t. to the apv ($\{\{A\}\}$ is smaller than $\{\{A, B\}\}$, see below), nor w.r.t. communication (sending go is not optimal), nor is it simple ($\delta(A)$ has a conditional).

Note that in Example 7, without any doubt, A participated in the violation of the property. Thus, in principle, it is sound to blame A — no matter which deviation the parties actually ran, A was either the sole cause, or part of the cause for the violation. A suitable optimality notion can guarantee fairness, and, as we will see, even a form of completeness.

Optimality ought to be weak enough to recognize uncertainty, e.g., both plausible deviations in Example 6, but strong enough to eliminate deviations that introduce causal relations in addition to those inherent to the protocol, as in the provocation example.

A. Sane optimality notions and accountability

We say a deviation δ explains a trace t if it produces a $t' \in traces(P\delta)$ s.t. $t =_V t' := t|_V = t'|_V$. We can now formalize the intuition above as follows: no matter which trace t we observe, even if resulting from a non-optimal deviation, the verdict (which is computed on $t|_V$) equals the apv *for all optimal deviations* that can explain t. Hence correctness holds for arbitrary traces, but the imaginary 'judge' considers only simple explanations for them. This definition can be simplified if the optimality notion is sane.

Definition 6 (sane optimality)**.** An optimality notion is *sane* if, for all protocols Π, (a) for all deviations δ there is an optimal deviation δ_o with $dom(\delta_o) \subseteq dom(\delta)$ and $\delta(p) \neq n(p)$ for all $p \in dom(\delta_o)$, s.t. for each $t \in traces(\Pi\delta)$, an optimal trace $t_o \in traces(\Pi\delta_o)$ of δ_o explains t, and (b) In addition, any optimal deviation explains some t produced by some (non-optimal) deviation.

Lemma 1. For any sane optimality notion, the following statements coincide.

1) For all δ, $t \in traces(P\delta)$ and optimal δ_o, $t_o \in traces(P\delta_o)$, $t =_V t_o \implies verdict(t) = apv(t_o, \delta_o)$.
2) For all optimal $\delta_o, t_o \in traces(P\delta_o)$, $verdict(t_o) = apv(t_o, \delta_o)$.

Note the difference between both formulations: the second considers only optimal deviations. The first, by contrast, provides a guarantee for all deviations; this guarantee, however, is that the judgment will be correct w.r.t. any optimal deviation that matches. If the optimality notion is sane, we can safely use the second, simpler formulation.

Proof. ($\Leftarrow$) We add the quantification over δ and t to bring (2) in the form of (1). We then weaken the statement by adding the antecedent of the implication. If $t =_V t_o$, then $verdict(t_o) = verdict(t)$ by congruence.
($\Rightarrow$) If $t =_V t_o$, then $verdict(t_o) = verdict(t)$ by congruence. Hence, δ_o is quantified over the set of optimal deviations that explain some trace produced by some deviation. By sanity, this is the set of all optimal deviations. □

Hence, we can use the simpler variant to capture this intuition.

Definition 7 (accountability)**.** We define accountability w.r.t. a sane optimality definition as in Def. 5, but quantify δ and t over the set of optimal deviations and traces.

Sanity also guarantees a form of fairness: any party blamed deviated indeed (but was not necessarily part of a cause).

Lemma 2 (weak fairness)**.** If *verdict* provides Π with accountability for a sane notion of optimality, then for all deviations δ and $t \in traces(P\delta)$,

$$\mathrm{S} \in verdict(t) \wedge p \in \mathrm{S} \implies p \in dom(\delta) \wedge \delta(p) \neq n(p).$$

Proof. For any optimal δ_o, let p st.t. $\delta_o(p) = n(p)$ or $p \notin dom(\delta_o)$. For any $\mathrm{S}' \supseteq \mathrm{S}$, if $t' \in traces(\Pi\delta_o|_{\mathrm{S}'})$ then $t' \in traces(\Pi\delta_o|_{\mathrm{S}'\setminus\{p\}})$. Hence, for any trace t_o, any $\mathrm{S} \in apv(t_o, \delta_o)$ that contains such a p contradicts the minimality requirement in Def. 4. By sanity and Lemma 1, for any deviation δ and trace t, there is a deviation δ_o and trace t_o with $t =_V t_o$ for which the above holds true. Hence $\mathrm{S} \in verdict(t)$ and $p \in \mathrm{S}$ implies $p \in dom$ and $\delta_o(p) \neq n(p)$. From $dom(\delta_o) \subseteq dom(\delta)$ follows $p \in dom(\delta)$ and thus $\delta(p) \neq n(p)$. □

VI. Verdict-optimality

The first option is to minimize the apv of each candidate deviation and resulting trace. Verdict order is reverse logical entailment between verdicts interpreted as propositional formulas.

Definition 8 (verdict order)**.** $\mathcal{S}_1 \leq \mathcal{S}_2$ if $\bigvee_{\mathrm{S}\in\mathcal{S}_2} \bigwedge_{p\in\mathrm{S}} p$ implies $\bigvee_{\mathrm{S}\in\mathcal{S}_1} \bigwedge_{p\in\mathrm{S}} p$.

If no violation occurs in the trace, the apv outputs the empty verdict $\emptyset$. As a propositional formula, this translates to $\bigvee_{\mathrm{S}\in\emptyset} \bigwedge_{p\in\mathrm{S}} p = \bot$. As there are no negative atoms in these formulas, $\emptyset \leq \mathcal{S}_2$ only if $\mathcal{S}_2 = \emptyset$. Hence $\emptyset$ is the bottom element of this order, while $\{\emptyset\}$ is the top element; however, this a posteriori verdict only arises if the normative behavior of the protocol may produce a violation by itself.

Verdict optimality solves the provocation problem, as A is implied by $A \wedge B$. Besides weak fairness (Lemma 2), it guarantees that no one is blamed who would not be blamed otherwise, because all deviations and contexts that are disregarded imply the verdict.

Furthermore, it provides a weak form of completeness. Recall that full completeness would again raise the provocation problem. Every disjunct of the weaker verdict can be understood as a group of agents 'working together' to produce the violation. For each disjunct, at least one representative appears in a disjunct of the actual verdict. This representative has an incentive to point out more subtle degrees of responsibility lest she takes the blame by herself.

Theorem 1 (weak completeness)**.** If $\mathcal{S}_2 \neq \emptyset$ then

$$\mathcal{S}_1 \leq \mathcal{S}_2 \iff \forall \mathrm{S}' \in \mathcal{S}_2 \exists \mathrm{S} \in \mathcal{S}_1 . \mathrm{S} \subseteq \mathrm{S}'.$$

(Proof in Appendix D.)

VII. Knowledge optimality

The key to addressing the provocation problem is to disregard deviations that introduce causal relations which are not part of the protocol. In contrast to verdict optimality, which solves the problem indirectly, the second notion we propose flat out forbids communication between deviating parties during the protocol run. It allows them, however, to distribute information before the protocol run, so that we can define an order on the knowledge parties share. We first relax Def. 3 so deviating parties may use names or variables that do not occur in the normative process, but in other deviating parties.

Definition 9 (relaxed deviation)**.** A *relaxed deviation* δ for a protocol $\Pi = (A, n)$, is a partial function from $\mathcal{A} \setminus \mathcal{T}$ to plain processes, s.t. $fv(\delta(p)) \subseteq \bigcup_{p'\in dom(\delta)} fv(n(p'))$ and $fn(\delta(p)) \subseteq \bigcup_{p'\in dom(\delta)} fn(n(p'))$ for every $p \in dom(\delta)$. It induces an instance of Π, which is defined as in Def. 3.

This gives us a measure of the information shared. We can now compare two deviations by comparing the information available to deviating parties, excluding the information they already possess by definition of the normative behavior, i.e., $fv(n(p))$ and $fn(n(p))$.

Definition 10 (knowledge order)**.** $\delta_1 \leq \delta_2$ if $dom(\delta_1) \subseteq dom(\delta_2)$, and for all $p \in dom(\delta_1)$,

$$fv(\delta_1(p)) \setminus fv(n(p)) \subseteq fv(\delta_2(p)) \setminus fv(n(p)) \text{ and}$$
$$fn(\delta_1(p)) \setminus fn(n(p)) \subseteq fn(\delta_2(p)) \setminus fn(n(p)).$$

To preserve the requirement of Def. 9 that only deviating parties share information, we modify the restriction operator: $d|_\delta(p)$ is undefined if $p \notin \mathrm{S}$, and defined $\nu\vec{n}'.\delta(p)\sigma$ where σ and the sequence of names $\vec{n}'$ are chosen such that names or variables that become unavailable due to the restriction are assigned fresh names, or structurally similar terms with fresh names.[4] We substitute, e.g., sk_A and pk_A by sk_{dummy} and $pk(sk_{dummy})$ if the active substitution in the protocol mapped pk_A to $pk(sk_A)$. This preserves the requirements of Def. 9 and captures the intuition that removing any party from the deviation also removes information that only this party could have shared.

Definition 11 (knowledge-optimal)**.** A relaxed deviation δ and $t \in traces(\Pi\delta)$ are knowledge-optimal if $p_1, p_2 \in dom(\delta) \implies (p_1, p_2) \notin ctl(t)$, and δ is knowledge-order minimal.

Relaxed deviations are complete in the sense that any trace can be reproduced, except for the communication between deviating parties, as it is forbidden.

[4]Formally, we assume a bijection ρ between the set of names occurring in $fn(\delta(p)) \cup fn(fv(t)\sigma_A) \setminus \bigcup_{p'\in dom(\delta)\cap\mathrm{S}} fn(n(p'))$ (for σ_A the active substitutions in scope of $\cdot_p$ in $\Pi = (A, n)$) and a sufficiently large set of fresh names. Then σ substitutes every name $n \in fn(\delta(p)) \setminus \bigcup_{p'\in dom(\delta)\cap\mathrm{S}} fn(n(p'))$, not available due to the restriction anymore by a (unique) fresh name according to ρ, and every variable $v \in fv(\delta(p)) \setminus \bigcup_{p'\in dom(\delta)\cap\mathrm{S}} fv(n(p'))$, by ρ applied to $\sigma_A(v)$.

Lemma 3 (relaxed deviations completeness)**.** For any protocol Π, deviation δ and $t \in traces(\Pi, \delta)$, there is a relaxed deviation δ_r with $t_r \in traces(\Pi\delta_r)$ s.t.

1) both traces are equal, except for communication between deviating parties, i.e., $t_r =_E t|_{\not\in dom(\delta_r)}$, for $\not\in \mathrm{S} := \mathcal{A} \uplus \mathcal{A} \times \mathrm{Terms} \uplus \{(p_A, p_B, m) \mid p_A, p_B \notin \mathrm{S}\}$,
2) deviating parties do not communicate in the relaxed trace, i.e., $p_1, p_2 \in dom(\delta_r) \implies (p_1, p_2) \notin ctl(t_r)$,
3) $dom(\delta_r) \subseteq dom(\delta)$ and
4) $\delta(p) \neq n(p)$ for all $p \in dom(\delta_r)$.

(Proof in the extended version [33, Appendix **??**] .)

This lemma holds because communication is not authenticated. Relaxed deviations can share necessary secrets beforehand. In the extreme case, one deviating party can act on behalf of all others.

VIII. Separating verdict-optimality, knowledge-optimality and simple deviations

In the following example, knowledge-optimal accountability can be achieved, while verdict-optimal accountability cannot. The class of these examples is characterized by one or more parties that appear in different subsets of the apv of some trace. In this case, deviating parties can introduce new causal dependencies even in verdict-optimal deviations.

Example 8 (2-out-of-3 vote, [13, Section IV])**.** For a successful attack, two out of three servers A, B, C need to validate a compromised certificate for a trusted party T. All of them deviate by accepting it, which is publicly visible.

In this case, the correct verdict is $(A \wedge B) \vee (B \wedge C) \vee (A \wedge C)$. Assume, however, that any of the three servers, e.g., C, communicates with the two others, A and B, and decides to deviate only if *they* deviate, using some arbitrary communication protocol. For such a deviation δ and trace t, the apv would yield $apv(t, \delta) = \{\{A, B\}\}$, as there is a trace t' with $t' \in traces(\Pi\delta|_{\{A,B\}})$ and $\neg\varphi(t')$, but no such trace for $S = \{A, C\}$ or $S = \{B, C\}$. As long as the communication between A, B and C is hidden, we could make the same argument for B or A in place of C. Hence, the protocol described in Example 8 is not accountable w.r.t. verdict-optimal deviations. By contrast, deviating parties cannot communicate in any knowledge-optimal deviation that explains the observed trace. Moreover, we know that all three parties are deviating, since they observably validated the compromised certificate. In the following example, however, A, B and C convey their intentions via a side channel that T provides.

Example 9 (2-out-of-3 vote with tally)**.** We modify Example 8 so that T transmits the current tally to all servers. A and B validate the certificate blindly, but C waits and only validates it if it receives a tally of two.

In this case, again, the apv is $\{\{A, B\}\}$, but we cannot distinguish this from the scenario where all three parties validate the certificate no matter the current tally. Thus no verdict function can achieve knowledge-optimal accountability (in this example).

IX. Simple deviations and the soundness of the centralized-adversary setting

The previous example showed that causal links between deviating parties can be introduced via indirect communication. In this section, we further restrict deviations to simple deviations, i.e., relaxed deviations mapping to processes without conditionals. We will see that accountability w.r.t. simple deviations equals accountability in the centralized-adversary setting, which gives us an interpretation of the latter.

In the centralized-adversary setting, deviating parties are not explicitly modeled. Instead, a single network adversary is modeled, in the form of a labeled reduction transition system that non-deterministically chooses the message a listening party receives. To represent the deviation of a party, we thus substitute deviating parties by a process that outputs all their secrets to the public and, therefore, to this network adversary.

Analogous to the set of traces $traces(A)$, we define the set of traces $traces^{cent}(A)$ for a closed extended process A. It comprises all sequences of labels in some reduction sequence according to the additional rules in Figure 4. Observe that these match the rules for labeled semantics of the applied-pi calculus [3, Sec. 4.3]. The only difference is that processes and labels are additionally tagged with party identities (or `context` for the centralized adversary).[5]

The centralized-adversary setting does not provide individual parties with the ability to change their behavior. We therefore transform the protocol into a closed extended process that gives the network adversary control over deviating parties. If the network adversary chooses to obtain these secrets, we consider this party *corrupted*, which is visible in the trace as an event.

Definition 12 (protocol transformation)**.** Let $[\![\Pi]\!] := \Pi\delta_{cor}$ where the deviation δ_{cor} has domain $\mathcal{A} \setminus \mathcal{T}$ and outputs each agent's secrets $\{x_1, \ldots, v_n\} = fv(n(p))$ and $\{n_1, \ldots, n_m\} = fn(n(p))$ as follows:

$$\delta_{cor}(p) := (\mathsf{event}\ corrupt(); \mathsf{out}(x_1); \ldots; \mathsf{out}(x_n); \mathsf{out}(n_1); \ldots; \mathsf{out}(n_m)) \quad \mid n(p)$$

The deviating parties are now those that the centralized adversary corrupted. Hence, we define

$$corrupted(t) = \{p \in \mathcal{A} \setminus \mathcal{T} \mid (p, corrupt()) \in t\},$$

and adapt Definitions 5 and 4.

Definition 13 (accountability in the centralized-adversary setting)**.** In the centralized setting, we say $verdict$ provides Π with accountability for φ, written $\Pi, verdict \vdash^{cent} \varphi$, if

$$\forall t \in traces^{cent}([\![\Pi]\!]).\ verdict(t) = apv_{\Pi,\varphi}(t).$$

[5] In the proofs that follow, we will refer to the 2016 version of this paper with the full version of the proofs [3]. Specifically, we will assume the correctness of the lemmas in [3, Appendix A-B]. They establish the soundness of a semantics on partial normal forms (see Appendix E). Our proofs follow Abadi, Fournet and Blanchet's proof that observational equivalence is labeled bi-similarity.

IN	$(\mathsf{in}(x).P)_{p_B} \xrightarrow{(\texttt{context},p_B,m)} P\{^m/_x\}_{p_B}$
OUT	$(\mathsf{out}(m).P)_{p_A} \xrightarrow{(p_A,\texttt{context},x)} P_{p_A}\{^m/_x\}$ if $x \notin fv(\mathsf{out}(m).P)$
CONTEXT-EVENT	$0_{p_A} \xrightarrow{(\texttt{context},m)} 0_{p_A}$
SCOPE	$\nu u.A \xrightarrow{\alpha} \nu u.A'$ if $A \xrightarrow{\alpha} A'$ and u does not occur in α
PAR	$A \mid B \xrightarrow{\alpha} A' \mid B$ if $A \xrightarrow{\alpha} A'$ and $bv(\alpha) \cap fv(B) = \emptyset$
STRUCT	$A \xrightarrow{\alpha} A'$ if $B \xrightarrow{\alpha} B'$ and $A \equiv B$ and $A' \equiv B'$

Fig. 4. Reduction rules for labeled semantics, in addition to Figure 3. We define $bv((p, \texttt{context}, x) = x$ and otherwise empty.

We define $apv_{\Pi,\varphi}(t)$ analogous to Def. 4:

$$\{\mathrm{S} \mid \neg\varphi(t) \text{ and } \mathrm{S} \subseteq corrupted(t) \text{ and S is minimal s.t. } \exists t' \in traces^{cent}([\![\Pi]\!]).corrupted(t') = \mathrm{S} \wedge t =_{\mathrm{S}} t' \wedge \neg\varphi(t')\}.$$

For accountability w.r.t. simple deviations, or shorter, simple accountability, we use the following notation:

$$\Pi, verdict \vdash^{simp} \varphi \iff \text{for all simple } \delta_r \text{ and } t_r \in traces(\Pi\delta_r).verdict(t_r) = apv(t_r, \delta_r).$$

In the first step, we show the completeness of the centralized adversary w.r.t. simple deviations. We apply a transformation to the labels because, in the centralized-adversary setting, the identities of deviating parties disappear from the trace. Let $t\{^S/_{\texttt{context}}\}$ denote t but with each element of form $t_i = (p_A, p_B, m)$ substituted by $(\texttt{context}, p_B, m)$ or $(p_A, \texttt{context}, m)$ if $p_A \in \mathrm{S}$ or $p_B \in \mathrm{S}$. Each element of form $t_i = (p_A, m)$ with $p_A \in \mathrm{S}$ is substituted by $(\texttt{context}, m)$. All $t_i = p_A \in \mathrm{S}$ are removed.

Lemma 4 (completeness centralized-adversary setting)**.** For all $t_r \in traces(\Pi, \delta_r)$ and δ_r relaxed, there is $t_c \in traces^{cent}([\![\Pi]\!])$ such that $t_c = t_r\{^{dom(\delta_r)}/_{\texttt{context}}\}$ and $dom(\delta_r) = corrupted(t_c)$. (Proof in Appendix F.)

Even if we only wanted to show the soundness of the centralized-adversary setting, we would need the soundness of traces in this setting to guarantee the minimality of verdicts.

Lemma 5 (soundness centralized-adversary setting)**.** For all $t_c \in traces^{cent}([\![\Pi]\!])$ there is $t_r \in traces(\Pi, \delta_r)$ with δ_r simple such that $t_c = t_r\{^{dom(\delta_r)}/_{\texttt{context}}\}$ and $dom(\delta_r) = corrupted(t_c)$. (Proof in the extended version [33, Appendix **??**] .)

The following lemma is necessary to relate the apvs in both settings. If two traces are related in the decentralized setting, the traces we get from Lemma 4 are related, too.

Lemma 6. Let $t_c, t_c' \in traces^{cent}(A)$, $t_r \in traces^{simp}(\Pi, \delta)$ and $t_r' \in traces^{simp}(\Pi, \delta')$. Assume that that $\mathrm{S} = dom(\delta) = corrupted(t_c)$ and $\mathrm{S}' = dom(\delta') = corrupted(t_c')$. If $t_r =_{\mathrm{S}'} t_r'$, $t_c = t_r\{^{\mathrm{S}}/_{\texttt{context}}\}$ and $t_c' = t_r'\{^{\mathrm{S}'}/_{\texttt{context}}\}$ then $t_c =_{\mathrm{S}} t_c'$.

Proof. We show that any two elements of t_r and t_r' that are matched by $=_{\mathrm{S}'}$ are either removed by the transformation or they are still corresponding after this substitution. Hence $r_c(t_c, t_c')$. *Case 1: Element in* S *or* $S \times$ Terms. The transformation $t_c = t_r\{^{\mathrm{S}}/_{\texttt{context}}\}$ removes all these elements from t_r and likewise for S′, t_c' and t_r'. As $\mathrm{S}' \subseteq \mathrm{S}$, those in S′ or $\mathrm{S}' \times$ Terms occur in neither t_c, nor in t_r. *Case 2: Elements in* $\mathrm{S}' \times \mathcal{A} \times$ Terms *or* $\mathcal{A} \times \mathrm{S}' \times$ Terms are modified to have `context` in the first or second position, depending on the case. □

The following lemma shows that matching traces from either setting — excluding communication between deviating parties — have the same apv. This is the key lemma to proving the equivalence of both settings.

Lemma 7 (similiar traces have same apvs)**.** If, for a simple relaxed δ_r, $t_r \in traces^{simp}(\Pi\delta_r)$ and $t_c \in traces^{cent}([\![\Pi]\!])$ such that $t_c =_{\not\supseteq \delta_r} t_r$,[6] then $dom(\delta_r) = corrupted(t_c)$ and $apv_{\Pi,\varphi,r}(t_r, \delta_r) = apv_{\Pi,\varphi,r}(t_c)$.

Proof sketch. For each S in the apv of one setting, we employ Lemmas 4 or 5 to produce a trace in the other setting. With Lemma 6, we relate this counterfactual trace to the actual trace. Now, using Lemma 4 or 5, whichever brings us back to the original setting, we show minimality. (Full proof in Appendix G.) □

Finally, we can show that both properties are equivalent.

Theorem 2. $\Pi, verdict \vdash^{cent} \varphi$ iff $\Pi, verdict \vdash^{simp} \varphi$.

Proof. Fix arbitrary Π, φ and $verdict$. *Soundness (⇒)*: First, assume $\Pi, verdict \vdash^{cent} \varphi$, but $\Pi, verdict \not\vdash^{simp} \varphi$. Then, there are δ_r and $t_r = traces(\Pi\delta_r)$ such that $verdict(t_r) \neq apv_{\Pi,\varphi,r}(t_r, \delta_r)$. By Lemma 4, there is $t_c \in traces^{cent}([\![\Pi]\!])$ such that $t_c =_{\not\supseteq \delta_r} t_r$. and $dom(\delta_r) = corrupted(t_c)$. From $t_c =_{\not\supseteq \delta_r} t_r$. we conclude that $verdict(t_c) = verdict(t_r)$. and hence from $\Pi, verdict \vdash^{cent} \varphi$ that $apv(t_c) = verdict(t_r)$. But from Lemma 7, we obtain $apv_{\Pi,\varphi,r}(t_r, \delta_r) = verdict(t_r)$ — contradicting the assumption.

Completeness (⇐) Second, assume $\Pi, verdict \not\vdash^{cent} \varphi$, but $\Pi, verdict \vdash^{simp} \varphi$. Then, there is $t_c \in traces^{cent}([\![\Pi]\!])$ such that $verdict(t_c) \neq apv(t_c)$. By Lemma 5, there is $t_r \in traces(\Pi, \delta_r)$ with δ_r knowledge-optimal, such that

[6]The relation $\not\supseteq S$ was defined in Def. 3.

TABLE I
OVERVIEW OF NOTIONS MAPPED OUT IN THIS PAPER.

	accountability	$\Longrightarrow$	verdict-opt. acc.	$\not\Longleftarrow$ $\overset{?}{\Longrightarrow}$	$\begin{pmatrix}\text{accountability}\\ \Downarrow \\ \text{knowledge-opt. acc.}\end{pmatrix}$	$\not\Longleftarrow$ $\not\Longrightarrow$	simple accountability ($\Longleftarrow$ acc.) $\Updownarrow$ acc. in the centralized-adversary setting
prerequisites	VC		NOINT		NOINT or NOIND		automated verification, widely applicable
weak fairness	✓		✓		✓		✓
weak completeness	✓		✓		× (see App. C)		× (see App. C)

VC... all communication visible NOINT... no intersections between subsets of verdict NOIND... no indirect communication between deviating parties

$t_c =_{\not\equiv\delta_r} t_r$ and $dom(\delta_r) = corrupted(t_c)$. The rest of the proof proceeds like in the first paragraph, but with the roles of t_r and t_c in reverse. □

X. CONCLUSION

We provided the first definition for accountability in the security setting that is not bound to a single corrupting adversary. Although the corrupting adversary is the default in protocol verification and cryptography, we found that it is not a sound approximation when considering accountability. This comes as a surprise, as this approximation is usually given little thought.

The provocation problem witnesses that, for realistic protocols, no distinct verdict can capture all possible explanations for a violation. The conclusion is that either unambiguity of the verdict needs to be dropped, the corruption model changed (e.g., to a single adversary controlling all deviating parties), or completeness weakened to some extent. In our opinion, unambiguity should not be given up. As Künnemann et al. [2] investigated the simplified corruption model already, the present work was dedicated to analyzing how to weaken completeness, and to understand which guarantees accountability in the centralized-adversary model effectively gives us, compared to the richer model we introduced here.

The most important result is a characterization of the centralized-adversary setting. Accountability in this setting is equivalent to the idea of a judge (Lemma 1) who assumes that deviating parties run programs without conditionals (Theorem 2). Prior work [2] gives us a verification method for accountability in the centralized-adversary setting. Now we understand what these results mean.[7]

[7]On a technical level, at least. What would motivate a judge to ignore branching deviations? A possible interpretation is based on 'defaults," i.e., 'assumptions about what happens when no additional information is given' [34]. They are *defeasible*, i.e., can be overwritten when new information arrives. Halpern and Hitchcock propose defaults to deal with difficult cases in causality; prior to that, default logics were introduced to formalize such inferences [35]. One could take the 0-Process as the default, and add inputs and outputs to account for messages that were observed. (This is, in fact, what happens in Lemma 5.) The default process for a given observation would not have conditionals, as there is no information about branches that were not taken. Indeed, judges frequently resolve cases that pose counterfactual questions by defaults, e.g., proof of habit or class-based presumptions about consumer behavior [36]. But note that critics of the approach rightfully point out 'under-constrained unclarities' [21] — the judge might have a different idea of what constitutes the default.

Other optimality notions do not assume deviations to be simple programs. All of them ensure that no party is ever blamed without actually deviating (Lemma 2); and that whenever something goes wrong, at least one party is blamed (Lemma 1). Verdict optimality even provides a weak form of completeness (Lemma 1) that guarantees that each verdict implicates at least one member of each collaborating group, hence incentivizing this 'representative' to supply additional information. An Example in Appendix C confirms that knowledge optimality does not afford this guarantee. However, it only expects deviating parties to share the minimum amount of knowledge before the protocol run, which seems a much more natural assumption than simple deviations.

Both these notions are therefore interesting in cases where our separation results do not apply. This includes protocols where apvs are always non-intersecting (e.g., access control [8], randomness generation [9]), in particular the case where a third party is to be held accountable [1], [6]. Even if verdicts may intersect, knowledge optimality applies when deviating parties cannot communicate indirectly, via trusted or non-deviating parties.

It is worth noting that all separating examples rely on causal dependencies introduced by deviating parties who exchange signals. These signals are never necessary for the attack itself. For many protocol tasks like the exclusion of parties in multi-round multiparty computation protocols or the detection of misbehaving trusted parties, such behavior is irrelevant. Until automated verification procedures for the decentralized setting become available, we thus advocate for the use of simple accountability when the accountability mechanism merely acts as a deterrent for misuse. First, it can be automatically verified in the centralized-adversary setting, providing trustworthy results. Second, we can guarantee weak fairness, hence, at a minimum, parties in the verdict failed to follow the protocol. Third, while completeness is limited, we know that at least one party is blamed. Nevertheless, as soon as the verdict has to be held to a higher standard — e.g., in court, where damages in civil cases can depend on the number of tortfeasors — then, at the very least, the underlying assumptions need to be understood.

REFERENCES

[1] R. Küsters, T. Truderung, and A. Vogt, "Accountability: Definition and relationship to verifiability," in *Proceed-*

ings of the 17th ACM Conference on Computer and Communications Security, ACM, 2010, pp. 526–535.

[2] R. Künnemann, I. Esiyok, and M. Backes, "Automated verification of accountability in security protocols," in *Computer Security Foundations Symposium*, 2019.

[3] M. Abadi, B. Blanchet, and C. Fournet, "The applied pi calculus: Mobile values, new names, and secure communication," *CoRR*, vol. abs/1609.03003, 2016.

[4] N. Papanikolaou and S. Pearson, "A cross-disciplinary review of the concept of accountability," in *Proceedings of the DIMACS/BIC/A4Cloud/CSA International Workshop on Trustworthiness, Accountability and Forensics in the Cloud (TAFC)*, 2011.

[5] 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, ACM, 2011, pp. 45–56.

[6] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *IEEE Symposium on Security and Privacy (S&P'98)*, IEEE Comp. Soc., 1998, pp. 86–99.

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

[8] 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, WPES 2005*, ACM, 2005, pp. 40–46.

[9] M. Backes, D. Fiore, and E. Mohammadi, "Privacy-preserving accountable computation," in *18th European Symposium on Research in Computer Security*, Springer, 2013, pp. 38–56.

[10] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07, ACM, 2007, pp. 175–188.

[11] R. Jagadeesan, A. Jeffrey, C. Pitcher, and J. Riely, "Towards a theory of accountability and audit," in *Computer Security–ESORICS 2009*, Springer, 2009, pp. 152–167.

[12] G. Gößler and D. L. Métayer, "A general framework for blaming in component-based systems," *Sci. Comput. Program.*, vol. 113, pp. 223–235, 2015.

[13] A. Datta, D. Garg, D. Kaynar, D. Sharma, and A. Sinha, "Program actions as actual causes: A building block for accountability," in *2015 IEEE 28th Computer Security Foundations Symposium*, IEEE, 2015, pp. 261–275.

[14] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *28th ACM Symp. on Principles of Programming Languages (POPL'01)*, ACM, 2001, pp. 104–115.

[15] B. Blanchet, "From Secrecy to Authenticity in Security Protocols," in *9th International Static Analysis Symposium (SAS'02)*, Springer, 2002, pp. 342–359.

[16] J. Y. Halpern and J. Pearl, "Causes and explanations: A structural-model approach — part 1: Causes," *CoRR*, vol. abs/1301.2275, 2013.

[17] J. Y. Halpern, "A modification of the halpern-pearl definition of causality," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, AAAI Press, 2015, pp. 3022–3033.

[18] D. Lewis, "Causation," *Journal of Philosophy*, vol. 70, no. 17, pp. 556–567, 1973.

[19] D. Hume, *An Enquiry concerning Human Understanding*, P. Millican, Ed., ser. Oxford World's Classics. Oxford University Press, 2007.

[20] R. Künnemann, "Sufficient and necessary causation are dual," *CoRR*, vol. abs/1710.09102, 2017.

[21] T. Blanchard and J. Schaffer, "Cause without default," in *Making a Difference*, Oxford University Press, 2014, pp. 175–214.

[22] R. Künnemann, D. Garg, and M. Backes, "Causality & control flow," in *4th Workshop on Formal Reasoning about Causation, Responsibility, & Explanations in Science & Technology*, 2019.

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

[24] G. Gössler and D. Le Métayer, "A General Trace-Based Framework of Logical Causality," in *FACS - 10th International Symposium on Formal Aspects of Component Software - 2013*, 2013.

[25] R. Dimitrova, R. Majumdar, and V. S. Prabhu, "Causality analysis for concurrent reactive systems," Tech. Rep., unpublished.

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

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

[28] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 1, 2010. (visited on 05/07/2020).

[29] S. Riahi, "Protocol accountability in the cryptographic setting," M.S. thesis, Saarland University, 2018.

[30] L. Fenton-Glynn, "A Proposed Probabilistic Extension of the Halpern and Pearl Definition of 'Actual Cause'," *The British Journal for the Philosophy of Science*, vol. 68, no. 4, pp. 1061–1124, 1, 2017. (visited on 05/07/2020).

[31] G. Gössler and J.-B. Stefani, "Causality Analysis and Fault Ascription in Component-Based Systems," p. 31,

[32] K. Morio, "A general definition of accountability in the cryptographic setting," Bachelor's Thesis, Saarland University, 2018.

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

[34] J. Y. Halpern and C. Hitchcock, "Graded Causation and Defaults," p. 41,

[35] V. W. Marek and M. Truszczynski, *Nonmonotonic Logic: Context-Dependent Reasoning*, ser. Artificial Intelligence. Springer-Verlag, 1993, ISBN: 978-3-540-56448-5. [Online]. Available: https://www.springer.com/gp/book/9783540564485 (visited on 05/08/2020).

[36] R. N. Strassfeld, "If . . .: Counterfactuals in the Law," p. 79,

APPENDIX

A. Operational semantics

In this section we define the operational semantics of our calculus for extended processes that do not contain holes, but in which the topmost process of any plain process is annotated with an *effectuating party*. Initially (cf. Def. 3), the topmost subprocess of every plain process inserted at a hole operator $\cdot_p$ is annotated with the party p, but scope restrictions are skipped. As we will see, only plain processes are annotated, but never scope restriction.

Structural equivalence is the smallest relation closed under α-conversion of names and variables, as well as application of evaluation contexts[8], such that:

PAR-0	$A_{p_A} \mid 0_{p_B} \equiv A_{p_A}$
PAR-C	$A_{p_A} \mid B_{p_B} \equiv B_{p_B} \mid A_{p_A}$
PAR-A	$A_{p_A} \mid (B_{p_B} \mid C_{p_C}) \equiv (A_{p_A} \mid B_{p_B}) \mid C_{P_C}$
NEW-0	$\nu u; 0_{p_A} \equiv 0_{p_A}$
NEW-C	$\nu u \nu v; 0_{p_A} \equiv \nu v \nu u; 0_{p_A}$
NEW-P	$A_{p_A} \mid \nu u; B_{p_B} \equiv \nu u; (A_{p_A} \mid B_{p_B})$ (if $u \notin fv(A) \cup fn(A)$)
ALIAS	$0_{p_A} \equiv \nu x; \{^M/_x\}$
SUBST	$\{^M/_x\} \mid A_{p_A} \equiv \{^M/_x\} \mid A'_{p_A}$ (where $A' = A\{^M/_x\}$)
REWR	$\{^M/_x\} \equiv \{^N/_x\}$ if $M =_E N$

(For brevity, A_p or B_p can stand for an unannotated process, in which case $p = \bot$.) Like in the applied pi calculus, we always assume that active substitutions are cycle-free, and that there is at most one active substitution for each variable in an extended process. Furthermore, there is exactly one active substitution when the variable is restricted. Later on, we will use variables to identify transmitted messages, hence we assume all variables to be unique from the start and define $P\{^m/_x\}$ as usual, but leave it undefined whenever P contains a subprocess of form $\mathsf{in}(x); P'$ to avoid dynamic renaming of variables. Using these rules, every closed extended process A can be brought into form [14]: $A \equiv \nu n_1; \ldots \nu n_m; (\{^{m_1}/_{x_1}\} \mid \ldots \mid \{^{m_l}/_{x_l}\} \mid P_1 \mid \cdots \mid P_k)$, where $P_1, \ldots, P_k$ are closed plain processes, i.e., all variables are bound or defined by an active substitution.

[8]Evaluation contexts are defined by the grammar

$$\langle C \rangle ::= \cdot \mid \nu n; \langle C \rangle \mid \nu x; \langle C \rangle \mid (A \mid \langle C \rangle) \mid (\langle C \rangle \mid A).$$

Internal reduction is the smallest relation on extended processes closed by structural equivalence and application of evaluation contexts such that:

THEN	$(\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q)_{p_A} \xrightarrow{p_A} P_{p_A}$ if $t_1 =_E t_2$
ELSE	$(\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q)_{p_A} \xrightarrow{p_A} Q_{p_A}$ if $t_1 \neq_E t_2$
COMM	$(\mathsf{out}(x); P)_{p_A} \mid (\mathsf{in}(x); Q)_{p_B} \xrightarrow{(p_A, p_B, x)} P_{p_A} \mid Q_{p_B}$
EVENT	$(\mathsf{event}\ m; P)_{p_A} \xrightarrow{(p_A, m)} P_{p_A}$

W.l.o.g., $p_A, p_B \neq \bot$, as structural equivalence preserves that the topmost non-ν position of any plain process remains annotated with an effectuating party. Along with ALIAS and SUBST, COMM permits the transmission of terms: Assume $x \notin fv(M) \cup fv(P)$, then

$$\begin{aligned}(\mathsf{out}(t); P) \mid (\mathsf{in}(x); Q) &\equiv \nu x; (\{^t/_x\} \mid (\mathsf{out}(x); P) \mid (\mathsf{in}(x); Q)) \\ &\to \nu x; (\{^t/_x\} \mid P \mid Q) \equiv P \mid Q\{^t/_x\}.\end{aligned}$$

Hence, we write $A \xrightarrow{(p_A, p_B, m)} B$ or $A \xrightarrow{(p_A, m)} B$ if $A \xrightarrow{(P_A, P_B, x)} B$ or $A \xrightarrow{(P_A, x)} B$ and x is in scope of an active substitution $\{^m/_x\}$ in A.

B. Policies for accountability

In the following, we show that such policies sometimes have to depend on the specific protocol to be useful; hence they are not truly protocol agnostic. Küsters et al. propose policies of the form $\alpha \implies p_1 \mid \cdots \mid p_n$ and define completeness as follows: if a trace matches α, the verdict should imply some p_i. Consider the scenario where each of the two parties A and B might violate a security property by deviating on its own, and assume the log always provides indication that this is the case. We would like to express that A and B shall be held accountable in case both deviate, as each deviation on their own would entail a violation. Let α match traces with said security violation. How could such a simple policy be expressed?

- $\alpha \implies A$ is too weak, as B's participation is disregarded in case only B deviates (and unfair towards A). Same for $\alpha \implies B$.
- $\alpha \implies A \wedge B$ is unfair to A or B in cases only one of them deviates.
- $\alpha \implies A \vee B$ permits uncertainty in the verdict as it would suffice to blame $A \vee B$ in case either deviates, but as A and B's behavior is visible, this policy is unnecessarily weak.
- $\alpha \implies A \mid B$: In this case, the verdict needs to imply either A or B, hence a verdict A would suffice even if both deviate. The same holds for the policy $\alpha \implies A \mid B \mid A \wedge B$.

Hence the only choice is to split α into two formulas, α_A and α_B, which capture traces where A, respectively B, misbehaves. Then α_A and α_B may intersect in case both

deviate and $\{\alpha_A \implies A, \alpha_B \implies B\}$ constitutes a policy that enforces actual completeness.

But at this point, the policy does not serve as a specification anymore — α_A and α_B describe the accountability mechanism itself, not the security property. For one, this means that the policy is protocol specific, i.e., not applicable to a class of protocols, e.g., all voting protocols, anymore. But most importantly, the policy validates the accountability mechanism with itself; hence the approach begs the problem.

C. Relaxed deviations are not weakly complete

The following example shows that, knowledge-optimal and simple accountability does not provide weak completeness like verdict-optimal accountability (Theorem 1).

Example 10. Consider a variant of the Whodunit-protocol (Ex. 6) where variables u, u' and v are available to A and B. The normative behavior of A is to send u to both B and T, and for B to forward whatever message it receives. Again, T checks for equality.

$n(A) =$out(u); out(u)
$n(B) =$in(x); out(x)
$n(T) =$in(x); in(y); if x=y then *0* else event *Unequal*()

In the following deviation, A sends two different messages to T and B, but B runs a deviation that forwards only the message u', precisely the message that A is not supposed to send.

$\delta(A) =$out(u); out(u')
$\delta(B) =$in(x); if x = u' then x else v

We consider the trace t where u arrives T, but u' at B and thus the event $Unequal()$ appears. The apv is $apv(t) = \{\{A\}, \{B\}\}$, because with $\delta|_{\{A\}}$, $n(B)$ would forward u' to T, which is also unequal to u. Likewise with $\delta|_{\{B\}}$; B would not forward the message it receives from A, but forward v instead.

The following simple and knowledge-optimal deviation explains the same trace. (We can make the same argument for δ_o only defined on B).

$\delta_o(A) =$out(u)
$\delta_o(B) =$out(v)

In this case, $apv(t_o) = \{\{B\}\}$. But B does not imply $A \vee B$. Or, in practical terms, the verdict for this deviation is not pointing to A at all, event though A's deviation by itself was sufficient to cause a violation.

D. Proof for Theorem 1

Proof.

$$\begin{aligned}
&\forall \mathrm{S}' \in \mathcal{S}_2 \exists \mathrm{S} \in \mathcal{S}_1 . \mathrm{S} \subseteq \mathrm{S}' \\
\implies &\forall \mathrm{S}' \in \mathcal{S}_2 \exists \mathrm{S} \in \mathcal{S}_1 . \bigwedge_{p \in \mathrm{S}'} p \implies \bigwedge_{p \in \mathrm{S}} p \\
\implies &\forall \mathrm{S}' \in \mathcal{S}_2 . \bigwedge_{p \in \mathrm{S}'} p \implies \bigvee_{\mathrm{S} \in \mathcal{S}_1} \bigwedge_{p \in \mathrm{S}} p \\
\implies &\bigvee_{\mathrm{S}' \in \mathcal{S}_2} \bigwedge_{p \in \mathrm{S}'} p \implies \bigvee_{\mathrm{S} \in \mathcal{S}_1} \bigwedge_{p \in \mathrm{S}} p \quad \text{if } \mathcal{S}_2 \neq \emptyset. \\
\iff &\mathcal{S}_1 \leq \mathcal{S}_2.
\end{aligned}$$

For the other direction, we show that $\exists \mathrm{S}' \in \mathcal{S}_2 \forall \mathrm{S} \in \mathcal{S}_1 . \mathrm{S} \not\subseteq \mathrm{S}'$ implies $\mathcal{S}_1 \not\leq \mathcal{S}_2$ by contradiction. Hence we assume that $\mathcal{S}_1 \leq \mathcal{S}_2$ and fix $\mathcal{S}_1$, $\mathcal{S}_2$ and S' such that

$$\mathcal{S}_1 \leq \mathcal{S}_2 \iff \bigvee_{\mathrm{S}' \in \mathcal{S}_2} \bigwedge_{p \in \mathrm{S}'} p \implies \bigvee_{\mathrm{S} \in \mathcal{S}_1} \bigwedge_{p \in \mathrm{S}} p \tag{1}$$

As $\mathrm{S}' \in \mathcal{S}_2$,

$$\bigwedge_{p \in \mathrm{S}'} p \implies \bigvee_{\mathrm{S} \in \mathcal{S}_2} \bigwedge_{p \in \mathrm{S}} p.$$

Hence by (1),

$$\bigwedge_{p \in \mathrm{S}'} p \implies \bigvee_{\mathrm{S} \in \mathcal{S}_1} \bigwedge_{p \in \mathrm{S}} p.$$

And thus there must be $\mathrm{S} \in \mathcal{S}_1$ such that

$$\bigwedge_{p \in \mathrm{S}'} p \implies \bigwedge_{p \in \mathrm{S}} p.$$

Hence $\mathrm{S} \subseteq \mathrm{S}'$, contradicting the assumption. □

E. Partial normal form

The partial normal form of an extended process A is an extended process of the form $\nu \vec{n}.(\{^{\vec{m}}/_{\vec{x}}\} \mid P)$ such that $(fv(P) \cup fv(\vec{m})) \cap \vec{x} = \emptyset$. [3, Appendix B.2] defines relations for structural equivalence, unlabeled and labeled reduction on partial normal forms, $\overset{\diamond}{\equiv}$, $\to_\circ$ and $\xrightarrow{\alpha}_\circ$. As our calculus is only a subset of the applied calculus with additional annotations, these relations and lemmas can be trivially transferred. We will assume them to hold in the proofs below.

F. Proof for Lemma 4

Proof. Fix any reduction sequence $A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n$. such that $t_r = (\alpha_1, \ldots, \alpha_n)$. Due to [3, Lemma B.9], w.l.o.g., $A_0 \to_\circ \alpha_1 \cdots \to_\circ \alpha_n A_n$. and all A_i in partial normal form. We will iteratively construct a reduction sequence for the centralized semantics such that, for any pair of positions i and $f(i)$ in these two sequences,

1) for t_c the trace in the centralized setting up to $f(i)$, $t_c = (\alpha_1, \ldots, \alpha_n)\{^{dom(\delta_r)}/_{\texttt{context}}\}$,
2) A_i has the form

$$A_i \overset{\diamond}{\equiv} \nu \vec{n}. \quad \Big\| \quad D_p \mid \quad \Big\| \quad H_p \mid \{^{\vec{m}}/_{\vec{x}}\}$$
$$p \in dom(\delta_r) \qquad p \in \mathcal{A} \setminus \mathcal{T} \setminus dom(\delta_r)$$

and

3) $B_{f(i)}$, for the same plain process H_p, names $\vec{x}, \vec{n}$ and messages $\vec{m}$

$$B_{f(i)} \overset{\diamond}{\equiv} \nu\vec{n}. \quad \Big\| \quad H_p \mid B_r\{^{\vec{m}}/_{\vec{x}}\} \uplus \sigma_H$$
$$p\in\mathcal{A}\setminus\mathcal{T}\setminus dom(\delta_r)$$

where B_r is an extended process which will be ignored. (The composition of two substitutions $\uplus$ is defined in [3, Appendix B.1].)

4) Both processes are closed, i.e., all free variables are defined by active substitutions.
5) The frame $\sigma = \{^{\vec{m}}/_{\vec{x}}\}$ comprises of $dom(\sigma_{init}) = \bigcup_{p\in\mathcal{A}} fn(n(p)) \cup fv(n(p))$.
6) The frame σ_H contains all messages send by honest parties to the context, i.e., for all $\{^m/_x\} \in \sigma_H$, $x \notin \vec{n}$, all m do not contain any name in $\vec{n}$, and for all D_p, any term m that occurs in D_p is equal to some $t\sigma \uplus \sigma_H$.

Base case:

$$A_0 = P[\delta_r] \equiv \nu\vec{n}. \quad \Big\| \quad H_p \mid \Big\| \quad D_p \mid \{^{\vec{m}}/_{\vec{x}}\},$$
$$p\in\mathcal{A}\setminus\mathcal{T}\setminus dom(\delta_r) \qquad p\in dom(\delta_r)$$

where $D_p = \delta_r(p)$ and $H_p = n(p)$. We start from

$$B_0 = [\![P]\!] \equiv \nu\vec{n}. \quad \Big\| \quad H_p \mid \Big\| \quad (\mathsf{event}\ \mathit{corrupt}(); \ldots) \mid \{^{\vec{m}}/_{\vec{x}}\}$$
$$p\in\mathcal{A}\setminus\mathcal{T} \qquad p\in\mathcal{A}\setminus\mathcal{T}$$

and reduce the corruption subprocesses for all $p_1, \ldots, p_m \in dom(\delta_r)$ using EVENT', OUT', SCOPE', PAR' and STRUCT'[9] until we reach a process $B_{f(i)}$: $B_0 \xrightarrow{p_1,\mathit{corrupt}()}_\circ \xrightarrow{p_1,\mathtt{context},x_1^1}_\circ \ldots \xrightarrow{p_1,\mathtt{context},x_1^{m_1}}_\circ \ldots \xrightarrow{p_1,\mathtt{context},n_1^1}_\circ \ldots \xrightarrow{p_1,\mathtt{context},n_1^{l_n}}_\circ \ldots \xrightarrow{p_k,\mathit{corrupt}()}_\circ \xrightarrow{p_k,\mathtt{context},x_k^1}_\circ \ldots \xrightarrow{p_k,\mathtt{context},x_k^{m_k}}_\circ \ldots \xrightarrow{p_k,\mathtt{context},n_k^1}_\circ \ldots \xrightarrow{p_k,\mathtt{context},n_k^{l_n}}_\circ \nu\vec{n}. \|_{p\in\mathcal{A}\setminus\mathcal{T}} H_p \mid \|_{p\in\mathcal{A}\setminus\mathcal{T}\setminus dom(\delta_r)} (\mathsf{event}\ \mathit{corrupt}(); \ldots) \mid \{^{\vec{m}\cdot\vec{s}}/_{\vec{x}\cdot\vec{x}_s}\}$.

Inductive case: Let $A_i \xrightarrow{\alpha}_\circ A_{i+1}$. We instantiate $B_{f(i)}$ from the IH. As both are closed, from [3, Lemma B.23 und B.24], we can distinguish the following cases for

$$A_{i+1} \equiv \nu\vec{n}. H'_{p_A} | H'_{p_B} | D'_{p_C} \quad \Big\| \quad H_p \mid \Big\| \quad D_p \mid \{^{\vec{m}}/_{\vec{x}}\}$$
$$p\in\mathcal{A}\setminus\mathcal{T}\setminus dom(\delta_r)\setminus\{p_A,p_B\} \qquad p\in dom(\delta_r)\setminus\{p_C\}$$

1) $H_{p_A} \xrightarrow{\alpha}_\circ H'_{p_A}$ (and $H'_{p_B} = H_{p_B}$, $D'_{p_C} = D_{p_C}$) In this case, $B_{f(i+1)=i+1}$ can perform the same transition.
2) $H_{p_A} \mid H_{p_B} \xrightarrow{\alpha}_\circ H'_{p_A} | H'_{p_B}$ via COM' (and $D'_{p_C} = D_{p_C}$). In this case, $B_{f(i+1)=i+1}$ can perform the same transition.
3) $D_{p_C} \xrightarrow{\alpha}_\circ D'_{p_C}$ via THEN' or ELSE' (and $H'_{p_A} = H_{p_A}$, $H'_{p_B} = H_{p_B}$). In this case, $B_{f(i+1)} = B_{f(i)}$.
4) $D_{p_C} \xrightarrow{p_C,m}_\circ D'_{p_C}$ via EVENT' (and $H'_{p_A} = H_{p_A}$, $H'_{p_B} = H_{p_B}$). From the IH, we can use OUT', SCOPE', PAR' and STRUCT', to reduce $B_{f(i)} \to_\circ (\mathtt{context}, m)B_{f(i+1)=f(i)+1}$.
5) $D_{p_C} = \mathsf{out}(m).D'_{p_C}$ and $H_{p_A} = \mathsf{in}(x).H'_{p_A}$ such that $A_i \xrightarrow{(p_C,p_A,m}_\circ$ and $\sigma_{i+1} = \sigma_i \cup \{^m/_x\}$ (and $H'_{p_A} = H_{p_A}$, $H'_{p_B} = H_{p_B}$). From the IH, we can use IN', SCOPE', PAR' and STRUCT', to reduce $B_{f(i)} \to_\circ (\mathtt{context}, p_A, m)B_{f(i+1)=f(i)+1}$.
6) $D_{p_C} = \mathsf{in}(x).D'_{p_C}$ and $H_{p_A} = \mathsf{out}(m).H'_{p_A}$ such that $A_i \xrightarrow{(p_C,p_A,m}_\circ$ and $\sigma_{i+1} = \sigma_i \cup \{^m/_x\}$ (and $H'_{p_A} = H_{p_A}$, $H'_{p_B} = H_{p_B}$). W.l.o.g., x does not occur in any D_p or H_p except D_{p_C}. From the IH, we can use OUT', SCOPE', PAR' and STRUCT', to reduce $B_{f(i)} \to_\circ (\mathtt{context}, p_A, m)B_{f(i+1)=f(i)+1}$ adding $\{^m/_x\}$ to σ'. This preserves the IH on σ_H and the process D_{p_C}.

□

G. Proof for Lemma 7

Proof. Fix t_c, t_r, δ_r and $S \subseteq \mathcal{A}$ s.t. $S \in apv(t_c)$. By Def. 4, there is $t'_c \in traces^{cent}([\![\Pi]\!])$ such that $corrupted(t'_c) = \mathrm{S}$ and $\neg\varphi(t'_c)$. For $\Pi = (A, n)$, we define $\Pi' = (A, n[\delta_r])$ where $n[\delta_r]$ is δ_r wherever δ_r is defined, and n otherwise. In Π', we define $dom(\delta_r)$ to be trusted. Observe that By Lemma 5, there is a simple relaxed deviation δ'_r and a trace $t'_r \in traces(\Pi, \delta_r)$ such that $t'_c =_{\not\approx\delta_r} t'_r$ and $dom(\delta'_r) = corrupted(t'_c) = \mathrm{S}$ Because $\neg\varphi(t'_c)$ and φ is congruent w.r.t. V, this means $\neg\varphi(t'_r)$. Because $S \subseteq dom(t_c)$, $t_c = |_{\mathrm{S}} t'_c$, $t'_c =_{\not\approx dom(\delta)\delta_r} t'_r$, and $t_c =_{\not\approx dom(\delta)\delta_r} t_r$, we can apply Lemma 6 to obtain $t_r = |_{\mathrm{S}} t'_r$. Therefore, and because δ'_r is simple, we have $\delta'_r = \delta_r|_{\mathrm{S}}$ (or there is a δ'_r such that this is the case and $t_r \in traces(P\delta'_r)$). Hence either $S \in apv(t_r, \delta_r)$, or there is a simple relaxed deviation δ^*_r and a trace $t^*_r \in traces(\Pi, \delta_r)$ such that $\neg\varphi(t^*_r)$, $dom(\delta^*_r) \subsetneq \mathrm{S}$. and $t'_r = |_{dom(\delta^*_r)} t^*_r$, and In the latter case, from Lemma 4, we have t^*_c with $t^*_c =_{\not\approx dom(\delta)\delta^*_r} t^*_r$ (hence $\neg\varphi(t^*_c)$) and $\mathrm{S}' = corrupted(t^*_c) = dom(\delta^*_r) \subsetneq \mathrm{S}$. From Lemma 6, we conclude that $t_c =_{\mathrm{S}'} t^*_c$. This, however, contradicts the minimality requirement of Def. 4 for $S \in apv(t_c)$. Hence $S \in apv(t_r, \delta_r)$ has to hold.

Any $S \notin apv(t_c)$ is either excluded by the minimality requirement, (in which case there is a strict subset $\mathrm{S}' \in apv(t_c)$, and, as argued before, $\mathrm{S}' \in apv(t_r, \delta_r)$, and thus $S \notin apv(t_r, \delta_r)$) or there is no t'_c with $t_c =_{\mathrm{S}} t'_c$ and $corrupted(t'_c) = \mathrm{S}$. In the latter case, we can apply the previous argument again: any simple relaxed deviation δ^*_r and trace $t^*_r \in traces(\Pi, \delta_r)$ such that $\neg\varphi(t^*_r)$, $t'_r =_{\mathrm{S}} t^*_r$ and $dom(\delta^*_r) = \mathrm{S}$ would imply the existence of t^*_c with $t_c =_{\mathrm{S}} t^*_c$ and $corrupted(t^*_c) = dom(\delta^*_r) = \mathrm{S}$, contradicting the assumption. Therefore, $S \notin apv(t_r, \delta_r)$, and thus $apv(t_c) = apv(t_r, \delta_r)$. □

[9] The variants of the rules in Figure 3 and 4 for the semantics on partial normal forms [3, p. 40].