Privacy-Preserving Accountable Computation

Michael Backes[1,2], Dario Fiore[1], and Esfandiar Mohammadi[2]

[1] Max-Planck Insitute for Software Systems, Saarbrücken, Germany
fiore@mpi-sws.org
[2] Saarland University, Saarbrücken, Germany
{backes,mohammadi}@cs.uni-saarland.de

Abstract. Accountability of distributed systems aims to ensure that whenever a malicious behavior is observed, it can be irrefutably linked to a malicious node and that every honest node can disprove false accusations. Recent work, such as PeerReview and its extensions, shows how to achieve accountability in both deterministic and randomized systems. The basic idea is to generate tamper-evident logs of the performed computations such that an external auditor can check the system's actions by mere recomputation. For randomized computations it is more challenging: revealing the seed of the pseudo-random generator in the logs would break the unpredictability of future values. This problem has been addressed in a previous work, CSAR, which formalizes a notion of accountable randomness and presents a realization. Although all these techniques have been proven practical, they dramatically (and inevitably) expose a party's private data, e.g., secret keys. In many scenarios, such a privacy leak would clearly be unaccepable and thus prevent a successful deployment of accountability systems.

In this work, we study a notion of privacy-preserving accountability for randomized systems. While for deterministic computations zero-knowledge proofs offer a solution (which is even efficient for some computations), for randomized computations we argue that efficient solutions are less trivial. In particular, we show that zero-knowledge proofs are incompatible with the notion of accountable randomness considered in CSAR if we aim at efficient solutions. Therefore, we propose an alternative definition of accountable randomness, and we use it as a building block to develop the new notion of privacy-preserving accountable randomized computation. We present efficient instantiations for interesting classes of computations, in particular for digital signature schemes as the arguably most important cryptographic primitive.

1 Introduction

In distributed systems, checking whether a node's operation is correct or faulty is a major concern. Indeed, faulty actions can occur for many reasons: a node can be affected by a hardware or software failure, a node can be compromised

J. Crampton, S. Jajodia, and K. Mayes (Eds.): ESORICS 2013, LNCS 8134, pp. 38–56, 2013.

by an attacker, or a node's operator can deliberately tamper with its software. Detecting such faulty nodes is often very difficult, in particular for large-scale systems.

Recent work proposed *accountability* as a paradigm to ensure that whenever an incorrect behavior is observed, it can be linked to a malicious node. At the same time, honest nodes gain the ability to disprove any false accusations. Examples of these accountability systems include PeerReview [15] and its extension [2]. The basic idea of PeerReview is that every user generates a tamper-evident log which contains a complete trace of the performed computations. Later, an auditor (in PeerReview any other node in the distributed system) can check the correctness of the user's operations by inspecting the logs, replaying the execution of the user using a reference implementation, and finally comparing its result. The above approach is however restricted to deterministic systems. Indeed, in order to enable the replay of a randomized computation one should publish the seed of the pseudo-random generator in the logs. Clearly, this would completely destroy the unpredictability of future pseudo-random values. This issue was addressed by CSAR, an extension of PeerReview [2]. More specifically, the main contribution in [2] is to formalize a notion of accountable randomness, called *strong accountable randomness*, and to present the construction of a pseudo-random generator satisfying this property. Informally, strong accountable randomness consists of the following requirements: (i) the pseudo-random generator generates values that look random, even to the party who computes them; (ii) it is possible to verify that the values were computed correctly; (iii) the unpredictability of future values (i.e., those for which a proof was not yet issued) does not get compromised; and (iv) the above properties are fulfilled even if a malicious party is involved in the seed generation.

While the approach of PeerReview and CSAR is very general and has been proven practical, these techniques have an inherent drawback: they inevitably expose a party's private data. In many scenarios such a privacy leak is unacceptable and might thus discourage the adoption of accountability systems. For instance, consider a company that runs its business using a specific software. There are many cases in which companies' tasks have to comply with legal regulations, and having a system which allows an auditor to check this compliance in a reliable way would be highly desirable. On the other hand, companies have a lot of data that they want to keep secret. This data might include, for instance, business secrets such as internal financial information, or secret keys for digital signatures or encryption schemes.[1]

In spite of its utter importance, the idea of providing accountability while preserving the privacy of the party's data has not been yet properly explored in previous work.

[1] While in principle such a problem can be solved by using generic secure multiparty computation techniques (SMPC) [10], all known SMPC protocols require the verifier to participate in the computation, which is infeasible in practice, whereas in our setting the verifier only participates in the verification by checking the tamper-resistant log, which is much better suited for practice.

1.1 Our Contribution

We address this important open problem in the area of accountability providing three main contributions:

- We formalize a notion of privacy-preserving accountability for randomized systems. At a high level, our notion requires that a user is able to produce a log that convinces an auditor of the correctness of (1) the outcome of a computation (e.g., that $y = P(x)$), and (2) the generated randomness. At the same time, the contents of the log neither compromise the secrecy of specific inputs of the computation nor the unpredictability of the randomness generated in the future. Our notion is defined in the UC framework, and thus allows for arbitrary composability.[2]
- We focus on efficient realizations of privacy-preserving accountability for randomized systems. We show that a construction can be obtained by using the non-interactive proof system by Groth and Sahai [13] which supports statements in the language of pairing-product equations, and a pseudorandom function, due to Dodis and Yampolskiy [9], which works in bilinear groups and is thus compatible with this language. With the above proof system we can characterize a variety of computations: efficient solutions exist for the case of algebraic computations with equations of degree up to 2, but also arbitrary circuits can be supported [12].
- We show interesting applications of privacy-preserving accountability for randomized systems to digital signatures. We present a signature scheme in which the signer can show that the secret key and the signatures are generated "correctly", i.e., by using accountable randomness. This essentially ensures that a signature has been created using a specific algorithm.

Our Contribution in Detail. In this section we give a high level explanation of the technical ideas and the approaches used in this paper.

Our notion and its relation with strong randomness. In the case of deterministic computations the notion of privacy-preserving accountability would essentially fall into the well-known application area of zero-knowledge proofs [11]. However, we model randomized computations: consequently we want that even the randomness is accountable, i.e., correctly generated. While such a notion, called strong accountable randomness, has already been introduced in [2], we show that it is *not* realizable without random oracles (see Section 3.1).

Recall that strong accountable randomness requires that the pseudorandomness of the generated values must hold even against the party who knows the seed. Clearly, this is a very strong property. A random oracle helps its realization as it essentially destroys any algebraic properties or relations that one may recognize in such values. But without the help of this "magic" tool, it is clear that the party computing the values knows *at least* how they were computed.

[2] We are aware that the UC framework has flaws. Our results, however, can be straightforwardly migrated to other simulation-based composability frameworks [23,18,16].

Our impossibility result left us with two opportunities: (1) either define privacy-preserving accountability for randomized computations in the strongest possible way (i.e., so as to imply strong randomness) but be aware that it would be realizable only using random oracles, or (2) define a slightly weaker version of accountable randomness. Although the first option would be preferable, a careful analysis revealed that its efficient realization is very unlikely. Indeed, any meaningful notion of privacy-preserving accountable computation fulfilling the properties we have in mind will need zero-knowledge proofs in order to be realized. At the same time, these proofs would have to involve a pseudo-random generator that satisfies strong randomness by using a hash function modeled as a random oracle. We are not aware of any hash function that allows for efficient zero-knowledge proofs and whose actual implementation maintains unpredictability properties close to the ideal ones of a random oracle (i.e., its use in a scheme does not fall prey to trivial attacks). This is why we decided to follow the second approach.

On realizing accountable signatures. While focusing on more specific applications of our accountability system, we asked how to efficiently prove statements that involve the randomness generated by our system. For instance, many cryptographic protocols rely on correctly distributed randomness, but such randomness usually cannot be revealed (thus CSAR is not a solution). In particular, this property is very interesting for digital signatures as it would allow for the accountability of this primitive, namely the signer could show that the secret key and the signatures are generated correctly (i.e., by using accountable randomness) and at the same time the signer does not leak such confidential data to the auditor.

Towards this goal, the technical challenge is that for the combination of Groth-Sahai proofs and our specific pseudo-random generator random values that need to be hidden can only be group elements.[3] We are not aware of any signature scheme, from the literature, in which *all* random values (e.g., the secret key and the randomness) can be computed using our pseudo-random generator. In this work we propose the construction of such a signature scheme which thus satisfies our notion of accountability.

2 Preliminary: The UC Framework

In this work, we formulate and prove our results in a composable, simulation-based model, in which the security of a protocol is obtained by comparison with an idealized setting where a trusted machine is available. More specifically, we use the UC framework [6]. Our results also apply to other simulation-based composability frameworks, such as IITM [18], RSIM [23], or GNUC [16].

We consider attackers that are global, static and active, i.e., an attacker that controls some parties and that controls the entire network. Such attackers are typically modelled in the UC framework by only considering protocols parties that have a designated communication tape for directly passing messages to

[3] In particular, every known pseudo-random function compatible with Groth-Sahai outputs group elements.

the attacker. Since the attacker controls the network, it can decide whether, in which order, and to whom network message are passed. Additionally every protocol party has a communication tape for directly passing messages to the so-called *environment*, a PPT machine that represents any user of the protocol, such as the web-browser or an operating system.

The security of a protocol is defined by comparing the execution of the protocol, i.e., of all protocol parties, with an idealized setting, called *ideal world*. The ideal world is defined just as the real world except for the existence of designated, incorruptible machines, called *ideal functionalities*. These ideal functionalities represent a scenario in which the same functionality is executed using a trusted machine to whom all parties have direct access. Formally, an ideal functionality directly communicates with the environment via so-called *dummy parties*, which forward all messages as instructed. This ideal functionality characterizes the leakage of the protocol and the possibilities of the attacker to influence the outcome of the protocol.

The security of a protocol π is defined by comparison with its corresponding ideal functionality $\mathcal{F}$ as follows: a protocol π *UC-realizes* an ideal functionality $\mathcal{F}$ if for all probabilistic polynomial-time (PPT) attackers $\mathcal{A}$ (against the protocol) there is a PPT attacker S (against the ideal functionality) such that no PPT machine (the environment $\mathcal{E}$) can distinguish an interaction with π and $\mathcal{A}$ from an interaction with $\mathcal{F}$ and S. A protocol π is considered UC-secure if it UC-realizes the corresponding ideal functionality.

For modeling setups, such as a PKI or a CRS, often ideal functionalities, such as $\mathcal{F}_{\text{CRS}}$, are used in description of the protocol. A setting in which both ideal functionalities and protocols occur is called a *hybrid world*. These ideal functionalities directly communicate with the protocol parties, since (formally) the protocol parties are part of the environment from the perspective of these ideal functionalities. These hybrid world can also be used to abstract away from cryptographic subprotocols, such as authenticated channels.

3 Defining Accountable Computation

In this section, we introduce a rigorous definition of accountable computation. As discussed in the introduction, this notion has to work for randomized systems, and thus have to guarantee accountable randomness. Towards this goal, we will first show that the previous notion of accountable randomness considered in [2] cannot be realized in the standard model. Then we will introduce our relaxed definition, for which we discuss efficient realizations in Section 4.

We consider a setting with a party Ve, called the auditor, that performs the audit and a computation party that performs the computation and, upon request, produces proofs that the computation has been correctly performed. Assuming an evaluation function Eval for computing results, an accountable computation scheme is a collection of three algorithms: Setup is run to generate the system's parameters that are distributed to every party and to the verifier; Prov is run by the party to prove statements about a computation and it produces a log; V is run by the verifier on input the log to check its correctness.

On $(\texttt{init})$ **from** $\mathcal{E}$
for honest P
draw random values
$r_1, \ldots, r_n \leftarrow \{0,1\}^\eta$
store $(\mathsf{rand}_1, \ldots, \mathsf{rand}_n) := (r_1, \ldots, r_n)$
send $(\texttt{init})$ to $\mathcal{A}$
output $(\texttt{initd})$ to $\mathcal{E}$

On $(\texttt{getrand}, i)$ **from** $\mathcal{A}$
send rand_i to $\mathcal{A}$

On $(\texttt{comp}, i, sid)$ **from** $\mathcal{E}$
for honest P
set $\mathsf{re} := (\mathsf{rand}_i)$
store $\mathsf{proofs}(sid, i) := (\mathsf{re}, 1)$
send (re, i, sid) to $\mathcal{A}$ and $\mathcal{E}$

On $(\texttt{comp}, i, sid)$ **from** $\mathcal{E}$ **for malicious P**
send $(\texttt{comp}, i, sid)$ to $\mathcal{A}$
wait for a response $(output, re, b, sid')$ from $\mathcal{A}$
store $\mathsf{proofs}(sid', i) := (re, b)$
output $(output)$ to $\mathcal{E}$

On $(\texttt{vr}, i, sid)$ **from** $\mathcal{E}$ **for honest Ve**
let $(\mathsf{re}, b) := \mathsf{proofs}(sid, i)$
if $b = 1$
then output $(\mathsf{rand}_i, i, 1)$ to $\mathcal{E}$
else output $(\mathsf{re}, i, 0)$ to $\mathcal{E}$

On $(\texttt{vr}, i, sid)$ **from** $\mathcal{E}$ **for malicious Ve**
send $(\texttt{vr}, i, sid)$ to $\mathcal{A}$
wait for a response m from $\mathcal{A}$
output m to $\mathcal{E}$

Fig. 1. The ideal functionality $\mathcal{F}_{\text{SR}}$ for strong randomness generation

For deterministic computations and for proofs that should not hide any secrets (e.g., decryption keys) previous work offers efficient solutions [15,14]. In the case of randomized computations, however, the computing party additionally needs to prove that the randomness has been honestly generated, e.g., in order to prove that signature key does not intentionally leave a trapdoor for malicious third parties. Therefore, randomized accountable computation needs a fourth algorithm INIT, that is run in a trusted set-up phase and in which the computing party gets a secret seed and the auditor a corresponding public seed.

Backes, Druschel, Haeberlen, and Unruh studied the problem of accountable randomness and introduced the notion of strong randomness [2]. The authors even presented an efficient construction that satisfies this property; however, their realization guarantees strong randomness only in the random oracle model. In the next section, we show that realizing their notion in the standard model is *impossible*.

NOTATION. In the description of the ideal functionalities and the protocol template, we use for persistently stored variables the font variable and for values the font *value*.

3.1 Strong Randomness Is Not Realizable

Backes, Druschel, Haeberlen, and Unruh define strong randomness by means of an ideal functionality $\mathcal{F}_{\text{SR}}$. This ideal functionality (formally explained in Figure

1) basically offers an interface for a computing party P to send commands to compute a pseudo-random generator, and $\mathcal{F}_{\text{SR}}$ offers an interface for auditors Ve to verify that these (pseudo-)random values are correctly distributed and unpredictably for the party that computes them. In addition, $\mathcal{F}_{\text{SR}}$ has an initialization phase, in which random values rand_i are drawn uniformly at random, and $\mathcal{F}_{\text{SR}}$ offers the attacker $\mathcal{A}$ a randomness oracle: upon a query $(\texttt{getrand}, i)$, $\mathcal{F}_{\text{SR}}$ responds with rand_i.

The Ideal Functionality $\mathcal{F}_{\text{sr}}$. Beside an initialization phase (via the command `init`), $\mathcal{F}_{\text{SR}}$ offers two commands `comp` and `vr`. If a party is malicious, $\mathcal{F}_{\text{SR}}$ allows the attacker to determine the behavior for these commands. For honest parties, upon $(\texttt{comp}, i, sid)$ the pseudo-random element with the index i is generated (and internally stored in $\mathsf{proofs}(sid, i)$). For honest parties, the flag b in $\mathsf{proofs}(sid, i)$ is set to 1 and for malicious parties, the flag is set to 0. Upon $(\texttt{vr}, i, sid)$, $\mathcal{F}_{\text{SR}}$ reads $\mathsf{proofs}(sid, i)$ and if $b = 1$ then it outputs the real random element and otherwise the stored element re.

For any (reasonable) two-party protocol Π, we show how the environment $\mathcal{E}$ can, in the standard model, easily distinguish whether it is communicating with Π and the real attacker $\mathcal{A}$ or $\mathcal{F}_{\text{SR}}$ and a simulator. Assume that the computing party P is compromised. Recall that $\mathcal{E}$ knows all secrets of P, in particular, any secret information used to compute the pseudorandom generator. We assume $\mathcal{A}$ to be the dummy attacker that simply forwards everything. $\mathcal{E}$ performs the following steps:

1. Send the command $(\texttt{comp}, i, sid)$ to P.
2. Since P is compromised, $\mathcal{A}$ has to answer for P. Since $\mathcal{A}$ is the dummy attacker, $\mathcal{A}$ forwards this duty to $\mathcal{E}$.
3. $\mathcal{E}$ computes the honest output (re, i) of that party P on its own, typically the output of a pseudo-random function on some seed and input i.
4. $\mathcal{E}$ sends the honestly computed output (re, i) as a response to $\mathcal{A}$.
5. $\mathcal{A}$ dutifully forwards the output (re, i) to the compromised P.
6. P sends the honestly computed output (re, i) over the network, i.e., to $\mathcal{A}$.
7. $\mathcal{A}$ simply delivers the message to Ve.
8. $\mathcal{E}$ sends $(\texttt{vr}, i, sid)$ to Ve and waits for a response $((re', i'), b)$
9. $\mathcal{E}$ outputs 1 if $(re, i) = (re', i')$ and $b = 1$; otherwise output 0

In the ideal setting, the attacker $\mathcal{A}$ will actually be the simulator. Now, if the simulator behaves differently from the attacker in steps 2, 5 or 7 (i.e., it does not let $\mathcal{E}$ compute the answer for P, does not forward the output (re, i) to the compromised P, or it does not deliver the message in step 7), then $\mathcal{E}$ can use this unexpected behavior to distinguish the two settings. Thus, the simulator has to act towards $\mathcal{E}$ as the dummy attacker (see step 5). At this point we have two possible cases for the answer of $\mathcal{F}_{\text{SR}}$ to the environment upon the command $(\texttt{vr}, i, sid)$: (i) either $b = 0$, or (ii) $b = 1$. In the case when $b = 0$, the environment will output 0 regardless of the value re'. If $b = 1$, recall that $\mathcal{F}_{\text{SR}}$ outputs $(re', i') = (\mathsf{rand}_i, i)$, where rand_i is uniformly chosen. Namely, $\mathcal{F}_{\text{SR}}$ replaces the value re sent by the simulator. Since rand_i is uniformly chosen, with

overwhelming probability we have that $\mathsf{rand}_i \neq re$. Hence, in the ideal setting $\mathcal{E}$ will output 0 with overwhelming probability. In contrast, in the real setting the environment always outputs 1 if $b = 1$ and 0 if $b = 0$.

We stress that in the random oracle model, this argument does not go through. Indeed, to compute re in step 3, $\mathcal{E}$ might have to query the RO. At that point the simulator could program the output of the RO such that it coincides with the uniformly chosen rand_i[4]. The main problem with this notion of strong randomness is realizing it against such a strong distinguisher (i.e., the environment) in UC. Since the seed of a pseudo-random function cannot be hidden from the environment, the latter can easily distinguish random values from the output of the pseudorandom function. We remark that the ideal functionality given in [2] is presented in a simplified setting where prover and verifier are the same machine. This can be done by assuming that the verifier is always honest (as verification is a public procedure). It is not hard to see that our counter-example works for this simplified setting as well. Indeed, we are not making any assumption on the honesty of the verifier.

3.2 Our Notion of Accountable Computation

In the standard model, it is not possible to realize strong randomness (see Section 3.1). The main problem is that the output of the pseudo-random generator has to be unpredictable even to the party that performs the computation. Unsurprisingly, such a result cannot hold in the standard model. Therefore, we weaken the definition of strong randomness in order to adapt it and make it realizable in the standard model. To do so, intuitively we require that the outputs of the pseudo-random generator be indistinguishable from random as long as the seed remains hidden. However, since our main goal is to provide accountability for the computations performed by the system, we directly integrate this (weaker) definition of strong randomness into a fully-fledged definition of accountable randomized computation.

Protocol Template for Real Accountable Computation. The core of an accountable computation scheme are four algorithms $(\textsc{Setup}, \textsc{Init}, \textsc{Prov}, \mathrm{V})$ that will be used by the parties P and Ve in a canonical protocol.

This protocol template assumes authenticated channels between party P and auditor Ve. This assumption corresponds to the common assumption that accountable systems have to maintain a tamper-evident record that provides non-repudiable evidence of all the actions that are sent via these authenticated channels. This authenticated channel is abstracted as an ideal functionality $\mathcal{F}_{\text{AUTH}}$ that guarantees that the network attacker cannot send messages on behalf of P. Typically, such an authenticated channel is realized using a PKI and by attaching a digital signature to every message[5]. Moreover, we introduce two set-up functionalities. The first set-up is a standard CRS functionality $\mathcal{F}_{\text{CRS}}$ that is needed

[4] Roughly speaking, this is the approach taken by the proof in [2].

[5] The functionality $\mathcal{F}_{\text{AUTH}}$ is standard, we do not present its definition here. We refer the interested reader to previous work [6].

for creating non-interactive zero-knowledge proofs. Technically, $\mathcal{F}_{\text{CRS}}$ runs the SETUP algorithm and distributes its output to all parties. The second set-up assumption models the initial trusted phase in which P receives a seed, for generating pseudorandom values, and VE receives a corresponding public information, which will enable VE to later check whether the pseudorandom values are generated correctly. This set-up assumption is modeled as a functionality $\mathcal{F}_{\text{PKIF}}$ that internally runs the INIT algorithm and accordingly distributes the result, i.e., the seed to P and the public information to VE. The goal of having $\mathcal{F}_{\text{PKIF}}$ is to ensure that the seed is generated truly randomly, even at a malicious party P. In practice this assumption can be realized in several ways, e.g., P and VE run a parallel coin tossing protocol, INIT is executed in a trusted hardware or in a phase of the protocol where P is guaranteed to behave honestly, or INIT is externally executed by a trusted entity who securely distributes the output. We stress that, even if not very efficient, this phase has to be run only once.

The computing party P is initialized before its first run (via `init`), and then it can be invoked (via `comp`) as a subroutine for computing programs (storing proofs about the execution) and publicly announcing the results. Moreover, P reacts to network requests (via `vr`) to prove statement about its announced results. The auditor VE is invoked by $\mathcal{F}_{\text{PKIF}}$ for the initialization of P (via `init`), and then can be invoked (via `vr`) as a subroutine to verify publicly announced results. Last, VE reacts to network announcements of P (via `cp`) that a result has been computed. The computing party P, upon `init`, queries both set-up functionalities $\mathcal{F}_{\text{CRS}}$ and $\mathcal{F}_{\text{PKIF}}$ in order to receive the public parameters and the seed for the pseudo-random generator. Upon an invocation $(\texttt{comp}, p, s, sid)$ with a program p and secret inputs s, P computes the program, adds a proof to the log and publicly announces the result. Upon a network message $(\texttt{vr}, re, p, sid)$ from the authenticated channel with VE, P outputs the corresponding proof, or an error message if such a proof does not exist.

The auditor VE, upon being called by an initialization message (`init`) from the seed generation, queries in turn $\mathcal{F}_{\text{CRS}}$ for the CRS and then stores all values. Upon a network message $(\texttt{prf}, re, p, cnt, sid)$ over the authenticated channel $\mathcal{F}_{\text{AUTH}}$ with P, VE stores the message and notifies the environement. Upon an invocation $(\texttt{vr}, re, p, cnt, sid)$, the auditor first asks via $\mathcal{F}_{\text{AUTH}}$ the computing party P for a proof, and then verifies this proof and outputs the result to the environment.

Ideal Functionality for Accountable Computation. The desired security properties for accountable randomized computation are captured by the ideal functionality described in Figure 2. The functionality offers the same interface to the environment as the protocol template and represents the "ideal" behavior of the protocol. In addition, however, the functionality explicitly allows the attacker to intercept messages, and it internally maintains a randomness function RA_{se} for modeling pseudorandomness.

The ideal functionality has interfaces to both the channel from the environment to P and the channel from the environment to VE. Therefore, we distinguish

On (`init`) **from** $\mathcal{E}$
for honest P
$se \xleftarrow{\$} \{0,1\}^\eta$
$\mathsf{se} := se$; $\mathsf{cnt} := 0$
send (`init`) to $\mathcal{A}$
output (`initd`, cnt)
to $\mathcal{E}$ from P

On $(\mathtt{comp}, p, s, sid)$
from $\mathcal{E}$ **for honest P**
if $p \in L_{\mathcal{R}}$ **then**
store $\mathsf{tc} := \mathsf{cnt}$
run $\mathsf{re} := \textsc{Eval}^{\mathrm{RA}_{\mathsf{se}}}(p, \mathsf{cnt}, s)$
$\mathsf{sta}(sid) := (\mathsf{re}, p, cnt)$
$\mathsf{wit}(\mathsf{re}, p, cnt, sid) := s$
$\mathsf{pe}(sid) := (\mathtt{prf}, \mathsf{re}, p, \mathsf{tc}, sid)$
$\mathsf{cnt} := \mathsf{cnt} + 1$
send $(\mathtt{prf}, \mathsf{re}, p, \mathsf{tc}, sid)$ to $\mathcal{A}$
P outputs $(\mathsf{re}, p, \mathsf{tc}, sid)$ to $\mathcal{E}$

On $(\mathtt{deli}, sid)$ **from** $\mathcal{A}$
let $(\mathtt{prf}, re, p, cnt, sid) := \mathsf{pe}(sid)$
store $\mathsf{sta}(sid) := (re, p, cnt)$
Ve outputs $(\mathtt{cp}, re, p, cnt, sid)$ to $\mathcal{E}$

$\mathrm{Ra}_{\mathsf{se}}$: When called on (cnt)
if P is honest **then**
$r \xleftarrow{\$} Rg(\mathrm{F})$; output r
else
$r := \textsc{Eval}(\mathrm{F}, cnt, \mathsf{se})$; output r

On $(\mathtt{vr}, re, p, cnt, sid)$ **from** $\mathcal{E}$ **for honest Ve**
if $(re, p, cnt) = \mathsf{sta}(sid)$ **then**
send $(\mathtt{vr}, re, p, cnt, sid)$ to $\mathcal{A}$
wait for a response $(\mathtt{deli}, s', sid)$
if P is honest
then $\mathsf{secr} := \mathsf{wit}(re, p, cnt, sid)$
else $\mathsf{secr} := s'$
if $(\mathsf{re}, \mathsf{pg}) = (re, p) \wedge re = \textsc{Eval}^{\mathrm{RA}_{\mathsf{se}}}(p, \mathsf{tc}, \mathsf{secr}, \mathsf{se})$
then output $(\mathsf{re}, p, 1)$ to $\mathcal{E}$
else output $(\mathsf{re}, \mathsf{pg}, 0)$ to $\mathcal{E}$

Fig. 2. The ideal functionality for accountable computation

from which of these channels an environment message comes and to which we output messages. Moreover, the functionality maintains internal (shared) data-structures, such as sta and wit which are used for verification, and se which is used for pseudo-random values.

Upon (`init`) for P, the functionality honestly draws a random seed and notifies the attacker that it has been initialized. Upon $(\mathtt{comp}, p, s, sid)$, the ideal functionality computes the program on the inputs, stores the secret inputs for later verification, and publicly announces the result. Upon a message $(\mathtt{deli}, sid)$ by the attacker, the statement is registered in pg and the environment is notified. Upon $(\mathtt{vr}, re, p, cnt, sid)$, the functionality recomputes the result with the stored witness. We stress that for malicious parties the attacker is allowed to give the witnesses for the statement in the `deli` message. Otherwise, the simulator does not work, because the real protocol does not reveal the proof earlier.

In contrast to the real protocol, for honest P the functionality returns truly random values as a result of $\mathrm{RA}_{\mathsf{se}}$, instead of the result of the pseudorandom function. This basically models that the pseudo-random generation should satisfy the usual notion of pseudorandomness, in which the challenger is always honest.

We stress that for dishonest parties P (and honest VE) our ideal functionality still guarantees that the PRF has been honestly computed. Malicious parties in the ideal model are canonically modeled by merely forwarding the input to the attacker and storing its results in the internal data-structures, such as sta.

4 Instantiations of Accountable Computation Schemes

Now that we have a clear definition of accountable randomized computation, we will show how it can be realized by means of suitable cryptographic tools. First, we describe below a generic paradigm to achieve this notion. However, since in the most generic case this generic construction may lead to rather inefficient instantiations, we will then show how to realize *efficient* accountable randomized computation for a significant class of computations.

A GENERIC CONSTRUCTION. The basic idea is to use UC-secure protocols for non-interactive zero-knowledge proofs, a perfectly binding commitment, and a pseudorandom function. Moreover, we assume the availability of ideal functionalities for the generation of the common reference string, the random sampling of a seed for the PRF, and for implementing authenticated channels (e.g., using signatures). At a high level, the generic scheme works as follows. In the setup phase, the parties ask for the common reference string for the NIZK proof system. Next, to initialize the system, every user invokes the ideal functionality in order to obtain a random seed se of a pseudorandom function F_{se}. It also samples random coins $open_{se}$, and computes a commitment $C = \text{COM}(se; open_{se})$, which is published in the authenticated log. The pair $se, open_{se}$ is instead maintained by the party. Later, whenever a party is asked to compute a function p on inputs s, it will compute $re = p(s)$ and will create a proof π using the NIZK proof system for the NP statement "$\exists s : output = p(s)$". To prove correctness of randomness generation, i.e., that $r = F_{se}(cnt)$, the user can use the same approach and generate a proof for the statement "$\exists(se, open_{se}) : r = F_{se}(cnt) \wedge C = Com(se; open_{se})$". The proof $\Pi = (p, re, \pi, cnt)$ is published in the log. Finally, the auditor can verify proofs by running the verification procedure of the NIZK proof system.

4.1 Useful Tools and Definitions

Before describing our efficient instantiation, here we introduce the algebraic tools and the cryptographic primitives that will be useful in our construction.

BILINEAR GROUPS. Let $\mathcal{G}(1^k)$ be an algorithm that on input the security parameter 1^k outputs a tuple $\mathsf{pp}_{BM} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ such that: p is a prime of at least k bits, $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of order p, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable and non-degenerate bilinear map.

The q-Decisional Diffie-Hellman Inversion (q-DDHI, for short) problem in $\mathbb{G}_1$ (same definition would hold in $\mathbb{G}_2$) is defined as follows.

Definition 1 (q-DDHI). *Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \xleftarrow{\$} \mathcal{G}(1^k)$, $g_1 \in \mathbb{G}_1$ be a generator, and $x \xleftarrow{\$} \mathbb{Z}_p$ be chosen at random. Let T be the tuple $(g_1, g_1^x, g_1^{x^2}, \ldots, g_1^{x^q})$, and Z be a randomly chosen element of $\mathbb{G}_1$. We define the advantage of an*

adversary $\mathcal{A}$ *in solving the* q-Decisional Diffie-Hellman Inversion *problem as* $\mathbf{Adv}_{\mathcal{A}}^{qDDHI}(k) = \left|\Pr[\mathcal{A}(p, T, g^{1/x}) = 1] - \Pr[\mathcal{A}(p, T, Z) = 1]\right|$, *where the probability is taken over the random choices of* $\mathcal{G}, x, Z$ *and the random coins of* $\mathcal{A}$. *We say that the* q*-DDHI Assumption holds in* $\mathbb{G}_1$ *if for every PPT algorithm* $\mathcal{A}$, *and for any* q *polynomial in* k, *the advantage* $\mathbf{Adv}_{\mathcal{A}}^{qDDHI}(k)$ *is negligible.*

Groth-Sahai proof system. Groth and Sahai [13] describes a way to generate efficient, non-interactive, witness-indistinguishable proofs for statements in the language $\mathcal{L}_{\mathsf{GS}}$ of so-called "*pairing product equations*". If $\{\mathcal{X}_i\}_{i=1}^m \in \mathbb{G}_1$ and $\{\mathcal{Y}_i\}_{i=1}^n \in \mathbb{G}_2$ are variables, and $\{\mathcal{A}_i\}_{i=1}^n \in \mathbb{G}_1$, $\{\mathcal{B}_i\}_{i=1}^m \in \mathbb{G}_2$, $a_{i,j} \in \mathbb{Z}_p$ and $t_T \in \mathbb{G}_T$ are constants, $\mathcal{L}_{\mathsf{GS}}$ is the language of equations of the following form:

$$\prod_{i=1}^{n} e(\mathcal{A}_i, \mathcal{Y}_i) \prod_{i=i}^{m} e(\mathcal{X}_i, \mathcal{B}_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{Y}_j) = t_T$$

The Groth-Sahai proof system can be instantiated in prime order groups by assuming its security based on either the SXDH or Decision Linear assumptions.

The main technique behind Groth-Sahai proofs is the use of specific commitment schemes that allow to commit to elements in $\mathbb{G}_1$ or $\mathbb{G}_2$. In particular, the proof system generates a common reference string which can be of two different and indistinguishable forms. When the CRS is instantiated for perfect soundness, the commitment is perfectly binding, whereas in the witness-indistinguishability setting the CRS leads to a perfectly hiding commitment. More importantly, the two modes of generation are computationally indistinguishable under the SXDH (resp. DLin) assumption, and both modes allow trapdoors that work as follows. In the perfectly binding setting, commitments have the form of ElGamal (resp. Boneh-Boyen-Shacham) ciphertexts, and the trapdoor is the decryption key, which thus allows to make the commitments *extractable*. In the perfectly hiding setting, instead, the trapdoor allows to equivocate the commitments, i.e., to create a commitment to some (random) value g_1^r, and to later open it to a different value g_1^x. These trapdoors are usually referred to as the extraction and simulation trapdoor respectively.

For lack of space, we refer the interested reader to [13] for a detailed and formal description of the Groth-Sahai proof system. Here we recall that such a scheme is defined by three algorithms (GS.Setup, GS.Prove, GS.Ver) that allow to, respectively, generate the parameters, create proofs and verify proofs. Moreover, for security, the system is also equipped with "extraction" and "simulation" algorithms (GS.ExtractSetup, GS.Extract, GS.SimSetup, GS.SimProve). In its basic instantiation, the Groth-Sahai scheme provides witness-indistinguishable proofs. However, Groth and Sahai interestingly show that for certain cases these techniques can be used to achieve zero-knowledge [13]. A significant case is the one in which all the equations being simultaneously satisfied have the constant value $t_T = 1$, the identity element in $\mathbb{G}_T$. Other statements have been shown to be modifiable in order to obtain zero-knowledge-friendly statements. We refer the interested reader to [13] for more details. For the sake of our work, we denote this subset of $\mathcal{L}_{\mathsf{GS}}$ that allows for zero-knowledge proofs as $\mathcal{L}_{\mathsf{GS-ZK}}$.

THE PSEUDORANDOM NUMBER GENERATION. As a tool for generating the randomness in our accountable computation we use the following pseudorandom function $F_s(c) = g_1^{\frac{1}{s+c}}$, which is also known as Boneh-Boyen weak signature [5], and Dodis-Yampolskiy PRF [9]. The function is proven pseudo-random under the q-DDHI assumption in $\mathbb{G}_1$, and for a domain D of size q where q is polynomial in the security parameter. We observe that the restriction on the domain's size is not a severe limitation in our setting as we will use the function in a stateful way to generate a sequence of values $F_s(1), F_s(2), \ldots$. The number of values is bounded by the the system's running time which is polynomial in the security parameter. More importantly for our application, the function is known to allow for efficient Groth-Sahai proofs. The idea of using zero-knowledge proofs to show the correctness of the outputs of a pseudorandom function is somewhat similar to the notion of simulatable verifiable random functions [7], with the only exception that in the latter case proofs do not need to be fully zero-knowledge. Belenkiy, Chase, Kohlweiss, and Lysyanskaya point this out [3] and propose a construction based on Groth-Sahai proofs.

4.2 An Efficient Instantiation of Accountable Computation

In this section we show how to realize accountable randomized computation for the language $\mathcal{L}_{\mathsf{GS-ZK}}$ of pairing product equations with zero-knowledge statements. It is worth noting that using $\mathcal{L}_{\mathsf{GS-ZK}}$ one can prove the simultaneous satisfiability of multiple algebraic equations whose degree is up to 2. In the description of our scheme we give an explicit description of the algorithms F and F.PROVE for the generation and the verification of the generated pseudorandom values. These algorithms are however a specific case of computations and proofs.

- SETUP(1^k): generate the description of bilinear groups $\mathsf{pp}_{BM} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ and the parameters $\mathsf{pp}_{GS} \xleftarrow{\$}$ GS.SETUP(pp_{BM}) of the Groth-Sahai proof system. Return $\mathsf{pp} = (\mathsf{pp}_{BM}, \mathsf{pp}_{GS})$.
- INIT(pp): as a seed, sample a random value $s \xleftarrow{\$} \mathbb{Z}_p$ and random opening $open_s$. The party keeps a secret key $fsk = (s, open_s)$ while a public verification key is $fpk = \text{COM}(g_2^s; open_s)$ is published to the log.
- PROV(pp, fsk, p, s): compute $re = p(s)$, run $\pi \xleftarrow{\$}$ GS.PROVE(pp_{GS}, St, w) where the statement St is created from the program p and the result re, whereas the witness is the secret input s. Output $\Pi = (p, re, \pi, cnt)$.
- F(pp, fsk, cnt): increment the counter $cnt \leftarrow cnt + 1$, and output $y = g_1^{\frac{1}{s+cnt}}$.
- F.PROVE(pp, fsk, cnt): proving the correctness of a pseudorandom value $y =$ F(pp, fsk, cnt) basically consists in creating a composable NIZK proof π for the language $\mathcal{L}_{PRF} = \{fpk, cnt, y : \exists s, open_s : fpk = \text{COM}(g_2^s; open_s) \wedge y = g_1^{\frac{1}{s+cnt}}\}$[6]. Output $\Pi = (\text{F}, y, \pi, cnt)$.
- V(pp, fpk, Π): parse Π as (p, re, π, cnt). Use the verification algorithm of Groth-Sahai to verify the proof π with respect to (public) values fpk, p, re, cnt.

[6] Belenkiy et al. show in [3] how to create such a proof using Groth-Sahai.

In terms of performances, the efficiency of the above instantiation heavily depends on the efficiency of the Groth-Sahai scheme. It is worth noting that although at the end our solution is not as efficient as CSAR, it is though the first providing such a strong privacy guarantee.

To prove the security of our accountable computation we will show that it realizes the ideal functionality of accountable randomized computation. In using the above instantiation in our protocol template we require the generation of different GS parameters pp_{GS} for every prover party. Generating different GS parameters, i.e., a different CRS, for every party avoids the need of being able to extract and simulate with the same CRS, which in turn allows us to use more efficient GS constructions. We stress that it is possible to use a strengthened GS proof system that allows for simultaneous simulation and extraction with the same CRS, and then to use only one CRS for all parties. However, since in our scenario we anyway assume the distribution of a public key for every prover, our restriction of using many CRS would not significantly weaken the set-up model.

Theorem 1. *Let Π be our protocol's template instantiated with the algorithms from Section 4.2, and let $\mathcal{F}$ be the ideal functionality from Figure 2. If the q-DDHI assumption holds in $\mathbb{G}_1$ and Groth-Sahai is secure, then Π securely UC-realizes $\mathcal{F}$.*

For lack of space, the proof of this theorem appears in the full version.

5 Using Verifiable Randomness Privately: Signatures

The previous section describes an accountable randomized computation for the language $\mathcal{L}_{\mathsf{GS-ZK}}$ of pairing product equations (of a certain form), and for a specific pseudorandom function $\mathrm{F}_s(x)$. It is worth noting that the generated (pseudo)randomness have a specific structure: the values are elements of the group $\mathbb{G}_1$. While in general one can use a suitable hash function in order to generate, e.g., binary strings out of group elements, such an arbitrary use of the randomness does not always allow for efficient zero-knowledge proofs. To be more concrete, if one wants to prove the correctness of a certain computation in which a value R generated using $\mathrm{F}_s(cnt)$ is one of the secret inputs, then R must be a variable in the language $\mathcal{L}_{\mathsf{GS-ZK}}$, i.e., R must be in $\mathbb{G}_1$.

Such a situation leaves us with an open question about the uses of the accountable randomness generated by our protocol. In this section, we address this problem and we propose an application to an important cryptographic primitive: digital signatures. In digital signatures, randomness is usually used to: (1) generate the secret signing key, (2) create the signature. If the randomness source is bad, the signature might be forgeable. Our scheme assumes a good randomness seed and given that seed proves that all signatures use "good" randomness.

5.1 An Accountable Signature Scheme

In this section, we tackle this problem and we propose a signature scheme that fits the setting of our accountable randomized computation, i.e., that of bilinear pairings. To achieve this goal, the faced technical challenge is that virtually

all existing constructions use either secret keys or random values that are "in the exponent". We solved this problem by proposing a new scheme which has the desired property, namely both the secret key and the randomness used in the signing algorithm are group elements. The proposed construction works within the accountable computation system. In particular, it uses the same pseudorandom generator and shares the same state.

The Security Model. Our signature scheme is stateful, in the sense that every message is signed with respect to a counter which gets incremented every time (in particular, the same counter is never re-used), and the signature is verified against the counter. For security, we consider the standard notion of *unforgeability under chosen message attack* in the stateful setting. This model considers an adversary that has access to a signing oracle and whose goal is to produce a forgery that either verifies against a "new" counter (i.e., a counter greater than the one in the system after the last query), or it verifies for an "old" counter (i.e., one for which a signature was obtained from the oracle) but for a message that is different from the one asked to the oracle.

Since our signature scheme is part of the accountability system (i.e., it shares the same parameters) we have to model the fact that an adversary may obtain additional information. For instance, it might ask for proofs about arbitrary statements. For this reason, we consider an extension of the unforgeability game, in which the adversary is granted access to an additional oracle $\mathcal{O}(\cdot)$ which can be either one of the algorithms $\textsc{Prov}(\mathsf{pp}, \mathit{fsk}, \cdot, s)$, $\mathrm{F}(\mathsf{pp}, \mathit{fsk}, \mathit{cnt})$, $\mathrm{F}.\textsc{Prove}(\mathsf{pp}, \mathit{fsk}, \mathit{cnt})$. We assume that F and F.Prove are computed on the next counter, whereas Prov is evaluated on a program p chosen by the adversary. For lack of space, a formal definition of our unforgeability experiment will appear only in the full version.

Our Construction. Before describing our construction, we give a high level description of our techniques. Our starting point is an idea, earlier proposed by Bellare and Goldwasser, for building signature schemes from zero-knowledge proofs [4]. Roughly speaking, Bellare-Goldwasser's scheme works as follows. The key generation consists of generating the seed s of a PRF and publishing its commitment C as the public verification key. Next, to sign a message m one computes the PRF on the message, $y = f_s(m)$, and proves in zero-knowledge that $y = f_s(m)$ and s is the same value in the commitment C.

In our case, the pseudorandom function is computed on a state, the counter, and thus we cannot apply it to an arbitrary message m. To solve this issue we create a signature on m by using the randomness $R = \mathrm{F}_s(\mathit{cnt}) \in \mathbb{G}_1$ and computing a value $\sigma = h^m \cdot R$, where h is also random value that is kept as the secret key. The actual signature is σ together with a zero-knowledge proof that σ is indeed created as $h^m \cdot \mathrm{F}_s(\mathit{cnt})$. The security relies on the soundness and zero-knowledge properties of the proof system, and the observation is that such value σ is essentially an information theoretic one-time MAC on m (if one assumes that h is random and so is every R).

More in detail, our construction works as follows. Let pp be the public parameters of the system consisting of a tuple $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \mathsf{pp}_{GS})$ where pp_{GS} are the parameters of a NIZK Groth-Sahai proof system.

- SigKeyGen(pp): use the pseudorandom function to generate $h \leftarrow \mathrm{F}(\mathsf{pp}, fsk, \mathsf{cnt}) \in \mathbb{G}_1$. Next, commit to h using random coins $open_h$, set $vk = \textsc{Com}(h, open_h)$ and $sk = (h, open_h)$.
- Sign(pp, fsk, sk, m). Let $m \in \mathbb{Z}_p \setminus \{0\}$ be the message, and let cnt be the system's counter for randomness generation. A signature on m is generated as follows. First, use the pseudorandom function to generate randomness $R \leftarrow \mathrm{F}(\mathsf{pp}, fsk, \mathsf{cnt})$. Next, compute $\sigma \leftarrow h^m \cdot R$, $C'_h = \textsc{Com}(h; open'_h)$ $C_R = \textsc{Com}(R; open_R)$, $C'_R = \textsc{Com}(R, open'_R)$, a composable NIZK proof π_1 for the statement $\exists(h, open'_h, R, open'_R) : \sigma = h^m \cdot R \wedge C'_h = \textsc{Com}(h, open'_h) \wedge C'_R = \textsc{Com}(R; open'_R)$, a composable NIZK proof π_2 that $\exists(g_2^s, open_s, R, open'_r) : C_R = \textsc{Com}(R; open_R) \wedge R = F_s(\mathsf{cnt})$, and composable NIZK proofs π_R and π_h proving that C_R and C'_R, and vk and C'_h commit to the same values. Output $\Sigma = (\sigma, C_R, C'_R, C'_h, \pi_1, \pi_2, \pi_h, \pi_R)$.
- SigVer($\mathsf{pp}, vk, m, \mathsf{cnt}, \Sigma$): use the verification algorithm of Groth-Sahai to verify proofs $\pi_1, \pi_2, \pi_R, \pi_h$.

Theorem 2. *If the Groth-Sahai proof system is secure, and the function $F_s(x)$ is pseudorandom, then the signature scheme is unforgeable.*

For lack of space, the proof appears only in the full version.

6 Related Work

Previous work proposed the use of accountability for several goals, such as to achieve real-world security [19], to incentivize cooperative behavior [8], to foster innovation and competition in the Internet [20,1], and to design dependable networked systems [24]. Systems have been built to provide accountability for both deterministic and randomized systems. In the previous section we already mentioned PeerReview [15] and its extension, CSAR [2]. Another example is CATS [25], a network storage service with strong accountability properties. The basic idea of CATS is to use a trusted publishing medium for publishing the logs and to ensure their integrity. The logs are then checked against a set of rules that describe the expected behavior of a node. Another system, repeat and compare [22], uses the accountability approach to guarantee content integrity in a peer-to-peer content distribution network built over untrusted nodes. Its basic idea is to use a set of trusted nodes that locally reproduce a random sample of the generated content and compare it to the one published by the untrusted nodes. Recently, another system, NetReview [14], successfully built upon the idea of PeerReview to enable the detection of faults caused by ISPs in the Border Gateway Protocol (BGP).

On the definitional side, Küsters, Truderung, and Vogt introduced a definition of accountability and compared it to the notion of verifiability [17]. They show that verifiability is weaker than accountability as the former does not require that a malicious party is always detectable. We notice that our definition implicitly assumes authenticated channels. Hence, it does not only capture verifiable computation, but also accountable computation.

The idea of generating accountable randomness is closely related to the notion of verifiable random functions (VRFs) [21], and simulatable VRFs [7]. In a nutshell, VRFs are pseudo-random functions that allow for publicly verifiable proofs about the correctness of the function's outputs. Moreover, all values for which a proof has not been issued are guaranteed to remain pseudorandom. Although this is intuitively the same requirement as in our case, there are a couple of differences due to some technical details. The difference mainly deals with the fact that our notion is simulation-based in a composability framework, and should not reveal any information about the seed, a property which is not necessarily captured by (simulatable) VRFs. To this extent, our techniques are related to the extension of simulatable VRFs proposed by Belenkiy, Chase, Kohlweiss, and Lysyanskaya [3], from which we borrow some of the technical ideas.

7 Conclusion and Future Work

In this paper we have investigated the notion of accountability for systems that execute randomized computations and want to keep the inputs of these computations private. We formalized a rigorous definition that models all the essential security properties, and we showed an efficient instantiation for interesting classes of computations based on techniques of the Groth-Sahai proof system. Furthermore, we proposed a digital signature scheme that enjoys the accountability properties of our system: the signer can convince an auditor that the secret signing key and the signatures are correctly generated (i.e., by using good randomness), and the auditor neither learns the signature key nor the randomness used for the signatures. For future work, it would be interesting to explore extensions of our scheme to provide accountability for other important cryptographic primitives, such as encryption, as well as to investigate efficient instantiations for richer classes of computations.

References

1. Argyraki, K., Maniatis, P., Irzak, O., Ashish, S., Shenker, S.: Loss and delay accountability for the internet. In: IEEE International Conference on Network Protocols, ICNP 2007, pp. 194–205 (October 2007)
2. Backes, M., Druschel, P., Haeberlen, A., Unruh, D.: Csar: A practical and provable technique to make randomized systems accountable. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2009 (2009)

3. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: Compact e-cash and simulatable VRFs revisited. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 114–131. Springer, Heidelberg (2009)
4. Bellare, M., Goldwasser, S.: New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 194–211. Springer, Heidelberg (1990)
5. Boneh, D., Boyen, X.: Short signatures without random oracles and the sdh assumption in bilinear groups. Journal of Cryptology 21, 149–177 (2008)
6. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. IACR Cryptology ePrint Archive, 2000:67 (2000)
7. Chase, M., Lysyanskaya, A.: Simulatable VRFs with applications to multi-theorem NIZK. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 303–322. Springer, Heidelberg (2007)
8. Dingledine, R., Freedman, M.J., Molnar, D.: Accountability. In: Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly and Associates (2001)
9. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005)
10. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, STOC 1987, pp. 218–229. ACM (1987)
11. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989)
12. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (2010)
13. Groth, J., Sahai, A.: Efficient noninteractive proof systems for bilinear groups. SIAM Journal on Computing 41(5), 1193–1232 (2012)
14. Haeberlen, A., Avramopoulos, I., Rexford, J., Druschel, P.: NetReview: Detecting when interdomain routing goes wrong. In: Proceedings of the 6th Symposium on Networked Systems Design and Implementation, NSDI 2009 (2009)
15. Haeberlen, A., Kuznetsov, P., Druschel, P.: PeerReview: Practical accountability for distributed systems. In: Proceedings of the 21st ACM Symposium on Operating Systems Principles, SOSP 2007 (2007)
16. Hofheinz, D., Shoup, V.: Gnuc: A new universal composability framework. IACR Cryptology ePrint Archive, p. 303 (2011)
17. Küsters, R., Truderung, T., Vogt, A.: Accountability: definition and relationship to verifiability. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, pp. 526–535 (2010)
18. Küsters, R.: Simulation-based security with inexhaustible interactive turing machines. In: Proc. 19th IEEE Computer Security Foundations Workshop, pp. 309–320 (2006)
19. Lampson, B.W.: Computer security in the real world. In: Proc. Annual Computer Security Applications Conference (December 2000)
20. Laskowski, P., Chuang, J.: Network monitors and contracting systems: competition and innovation. In: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM 2006, New York, NY, USA, pp. 183–194 (2006)

21. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th FOCS, New York, New York, USA, October 17-19, pp. 120–130 (1999)
22. Michalakis, N., Soulé, R., Grimm, R.: Ensuring content integrity for untrusted peer-to-peer content distribution networks. In: Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation, NSDI 2007, p. 11 (2007)
23. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. IEEE Symposium on Security and Privacy, p. 0184 (2001)
24. Yumerefendi, A.R., Chase, J.S.: Trust but verify: accountability for network services. In: Proceedings of the 11th Workshop on ACM SIGOPS European Workshop, EW 11, New York, NY, USA (2004)
25. Yumerefendi, A.R., Chase, J.S.: Strong accountability for network storage. In: 5th USENIX Conference on File and Storage Technologies (2007)