Towards Authenticity and Privacy Preserving Accountable Workflows

David Derler, Christian Hanser, Henrich C. Pöhls, Daniel Slamanig

Towards Authenticity and Privacy Preserving Accountable Workflows

David Derler[1], Christian Hanser[1], Henrich C. Pöhls[2], and Daniel Slamanig[1]

[1] IAIK, Graz University of Technology, Austria
{david.derler|christian.hanser|daniel.slamanig}@tugraz.at
[2] Institute of IT-Security and Security Law & Chair of IT-Security,
University of Passau, Germany
hp@sec.uni-passau.de

Abstract. Efficient and well structured business processes (and their corresponding workflows) are drivers for the success of modern enterprises. Today, we experience the growing trends to have IT supported workflows and to outsource enterprise IT to the cloud. Especially when executing (interorganizational) business processes on third party infrastructure such as the cloud, the correct execution and documentation become very important issues. To efficiently manage those processes, to immediately detect deviations from the intended workflows and to hold tenants (such as the cloud) accountable in such (decentralized) processes, a mechanism for efficient and accountable monitoring and documentation is highly desirable. Ideally, these features are provided by means of cryptography in contrast to organizational measures.

It turns out that variants of *malleable* signature schemes, i.e., signature schemes where *allowed modifications* of signed documents do not invalidate the signature, as well as *proxy (functional)* signature schemes, i.e., signature schemes which allow the *delegation of signing rights* to other parties, seem to be a useful tool in this context. In this paper, we review the state of the art in this field, abstractly model such workflow scenarios, investigate desirable properties, analyze existing instantiations of aforementioned signature schemes with respect to these properties, and identify interesting directions for future research.

1 Introduction

To efficiently handle frequently recurring processes within enterprises, it is advantageous to define standardized business processes. An ICT supported technical realization of a business process is usually denoted as a workflow [30]. Such a workflow can be seen as an abstract process, which defines a certain sequence of tasks as well as conditions on how participating entities have to complete these tasks. In such a context, workflows may span various departments within an enterprise or even various enterprises (interorganizational workflows).

The authors have been supported by EU H2020 project Prismacloud, grant agreement n°644962.

To always have an overview of the current state of concrete workflow instances and to be able to react to deviations from the defined workflows, it is important to document each step and to report it to some entity. Thereby, an inherent requirement is that these reports allow to verify whether delegatees acted within their boundaries and that each task can be attributed to a certain delegatee. This shall hold true especially if the process is interorganizational. In addition, it is desireable to automatically derive information, e.g., to issue warnings if certain constraints in a workflow are not met. Furthermore, it is often required that documentations of certain workflows are retained in an unforgeable recording for auditing or legal purposes. For example, the European data protection law requires an organization to document the usage of data [6]. However, the boundaries within each participating entity can act in a workflow might already be a sensitive business internal. Hence, it should not be disclosed to other parties (e.g., other enterprises in interorganizational workflows). Thus, an additional requirement is that the defined boundaries are not revealed to entities verifying a report, i.e., to ensure privacy, while still being able to check whether delegatees acted within their boundaries. We stress that this goal is in contrast to confidentiality of task reports. In particular, privacy requires that—even when the task reports are available in plain—the defined boundaries are not recoverable.

A suitable application is outsourcing inter- or intra-enterprise workflows to some environment that is not under full control, e.g., to the cloud. An automated process outsourced to the cloud may then run on behalf of the participants to carry out a task within such a workflow. A participating enterprise will be interested in the correctness of the workflow, the compliance with associated privacy requirements and to hold the cloud accountable. We note that especially in context of accountability there are significant efforts to provide and standardize frameworks for cloud accountability, e.g., as demonstrated within the A4Cloud project [44]. We note that we are interested in a more abstract view on workflows and cryptographic tools that allow to realize the aforementioned requirements.

1.1 Related Work

Besides [40,39,34], not much attention has been paid to cryptographically enforcing certain properties of workflows. Subsequently, we review the existing approaches and other related concepts.

In [40,39], the authors investigate traceability and integrity aspects of decentralized interorganizational workflow executions. This work focuses on preserving authenticity and integrity with respect to logical relations (AND, OR, XOR) among certain tasks in a priori defined workflows, while the concrete agents executing the workflow tasks do not need to be pre-specified (these could be dynamically chosen with the help of some discovery service). To do so, they use policy-based cryptography [4], where every agent gets issued credentials from some central authorities (specifying attributes that the agent satisfies). Then, for each workflow step a policy defines what needs to be satisfied for the execution of the respective task (basically the required decryption keys can only be obtained if the policy is satisfied). In addition they use group signatures to guarantee anonymity of honest agents, but support traceability of malicious ones.

In contrast, [34] allows to dynamically define those workflows during the workflow execution. That is, they map the workflow to a (dynamically extendable) tree, where each node in the tree is interpreted as one particular workflow task. Then, building upon the hierarchical identity based signature scheme in [35], one can build a hierarchy of signing keys (i.e., each node in possession of a signing key can issue signing keys for its child nodes). These signing keys are then used to sign some task-dependent information and, due to the hierarchical nature of the underlying primitive, this delivers an authentic documentation of the workflow execution regarding the logical relations among subsequent tasks.

Orthogonal to our goals of authenticity, accountability and privacy, variants of attribute-based encryption were used for cryptographically ensured access control with respect to some policy in [43,2,22]. Recent work [21], thereby, also considers the possibility to hide the access policy.

Somehow close to our goal is [28], but it does not target the enforcement of properties of workflows. However, the authors use malleable signatures to allow to remove (potentially confidential) information from signed data, while not influencing source authentication in service oriented architectures (SOAs). In their approach, workflow participants exchange signed data based on predecessor-successor relationships. This is not what we are looking for in this case.

Finally, the work done in this paper relates to data provenance, which deals with identifying the origins of data and also giving a record of the derivation [41]. More precisely, this work relates to the aspect of process documentation found in data provenance, i.e., the proposed solutions will allow to verify whether a certain workflow was carried out as intended. This and other aspects of data provenance have been surveyed and studied in the literature, for example in [47,42,23]. Our work may be considered as realizing some aspects of provenance with cryptographic guarantees, i.e., to ensure that any deviation from a planned workflow will be detectable and that each workflow participant can be held accountable for it's actions.

1.2 Motivation and Contribution

The few existing approaches to authenticity, accountability and privacy in workflows [40,39,34] rely on rather non-standard and often complex schemes. Given the importance of outsourcing computations and processes to cloud providers, it is thus an interesting challenge to look for simpler and more efficient solutions that rely on standard cryptographic primitives.

We propose two generic patterns to document the workflow executions, which can be instantiated using various different signature primitives. These patterns follow the well-known *delegation-by-certificate* approach from proxy signatures [37], and—in contrast to existing solutions—allow to obtain particularly efficient schemes which only make use of standard cryptographic primitives with multiple efficient instantiations. In addition to existing work, which only considers tasks from an abstract point of view, we also consider the outputs of tasks and their corresponding documentation (reports).[3] In this context, we discuss means to predefine the structure of reports to ease an automated processing and also cover related privacy issues. We develop a set of requirements for workflow documentation systems and analyze possible instantiations of our generic patterns from different types of signature schemes with respect to these requirements. Finally, we discuss open problems and future directions.

2 Preliminaries

Throughout the paper we require the notion of digital signature schemes, which we recall subsequently. A digital signature scheme (DSS) is a triple (KeyGen, $\mathsf{Sign}, \mathsf{Verify}$) of efficient algorithms. Thereby, KeyGen is a probabilistic key generation algorithm that takes a security parameter $\kappa \in \mathbb{N}$ as input and outputs a secret (signing) key sk and a public (verification) key pk. Further, Sign is a (probabilistic) algorithm, which takes a message $M \in \{0,1\}^*$ and a secret key sk as input, and outputs a signature σ. Finally, Verify is a deterministic algorithm, which takes a signature σ, a message $M \in \{0,1\}^*$ and a public key pk as input, and outputs a single bit $b \in \{0,1\}$ indicating whether σ is a valid signature for M under pk.

A digital signature scheme is required to be *correct*, i.e., for all security parameters κ, all $(\mathsf{sk}, \mathsf{pk})$ generated by KeyGen and all $M \in \{0,1\}^*$ one requires $\mathsf{Verify}(\mathsf{Sign}(M, \mathsf{sk}), M, \mathsf{pk}) = 1$. Additionally, for security one requires existential unforgeability under adaptively chosen-message attacks ($\mathsf{EUF\text{-}CMA}$) [24].

3 Workflow Model

In the following we align our notation largely with the one used in [34]. A *workflow* W comprises some central entity called the *workflow manager* (WM) who wants to outsource a workflow to some set A of entities denoted as *agents*. Thereby, every workflow can be decomposed into single atomic *tasks* $t_i \in T$, where every task is executed by some agent. For instance, task $t_i \in T$ may be executed by agent $A_j \in A$, which we denote by $A_j(t_i)$.

As it is common when modeling workflows (e.g., [29]), we define a workflow as a directed acyclic graph $W = (T, E)$, where each vertex $t_i \in T$ represents one particular task and edges $e_j \in E \subseteq T \times T$ represent task dependencies, i.e., a vertex $(t_u, t_v) \in E$ means that task t_v follows after the completion of task t_u. Now, we augment such a simple workflow by the following semantics and in the remainder of the paper we always mean such an augmented workflow when we speak of a workflow. Each vertex $t_i \in T$ with at least two outgoing edges (i.e., where outdegree $\deg^+(t_i) \geq 2$) is called a *split* and each vertex t_i with at least two incoming edges (i.e., where indegree $\deg^-(t_i) \geq 2$)) is called a *join*. Each split and join is associated with a logical type {AND,OR,XOR}. In case of an AND split all edges are executed in parallel; in case of an XOR split exactly one

[3] This could also be interesting in the context of data provenance.

edge must be executed; and in an OR split at least one edge needs to be executed. To illustrate this idea, we present an example of a simple workflow in Figure 1. For ease of presentation we label each outgoing edge with the respective type.

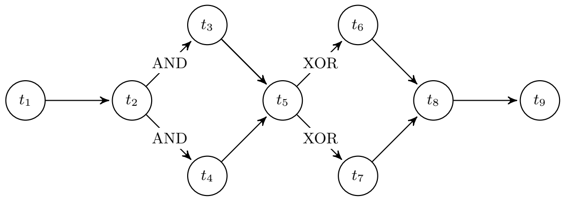

Fig. 1. A simple workflow example.

To distinguish between successful and unsuccessful workflow executions, we need the notion of a *trace*. A trace τ of a workflow is a sequence of tasks in the order of their execution and a trace is called *valid* if it is compatible with the workflow. Let us look at the example in Figure 1. For instance, the trace $\tau = (t_1, t_2, t_3, t_5, t_7, t_8, t_9)$ is invalid, but $\tau' = (t_1, t_2, t_3, t_4, t_5, t_7, t_8, t_9)$ is a valid trace.

Another issue that needs to be addressed is that not every agent may be allowed to execute every task. Consequently, we use *assignment* $\alpha(\tau)$ to denote the sequence indicating which agents have executed the respective task. For instance, we may have $\alpha(\tau') = (A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8)$. Furthermore, either the WM may specify which potential set of agents is allowed to execute each task (*static assignment*) or each agent may dynamically decide which agents may execute the subsequent task(s) (*dynamic assignment*). In case of a static assignment, we call an assignment $\alpha(\tau')$ valid if it is compatible with the restrictions set by the WM. We note that our above notation deviates from the one in [34] who only consider dynamic assignments. Also, in contrast to [34] who solely look at the tasks in a workflow from a very abstract level, we are also interested in properties of the outputs of the tasks and thus get a bit more concrete. Therefore, we introduce the notion of the documentation of one particular task in a workflow and denote it as the report of a task, or *report* in short.

Subsequently, we introduce desirable properties for the documentation of workflow executions. Firstly, the most crucial requirement in our setting is that reports are protected against unauthorized modifications. Recall, that we do not consider the orthogonal feature of providing confidentiality for workflow data.

Requirement 1. *The integrity of the reports needs to be ensured.*

Furthermore, it is required that only the workflow manager and the execution agent, which is actually performing a certain task, can produce a valid report.

Requirement 2. *For a particular task t_i, no one except the workflow manager and the agent(s) assigned to t_i is/are capable of creating task reports that are accepted by an auditor.*

In this context, it is also important that each report can be used to identify the respective execution agent (workflow manager), i.e., to ensure accountability.

Requirement 3. *The execution agent (workflow manager) that performs a certain task can be held accountable for its actions.*

However, as long as the work is done correctly, a delegator might want to account for the work of a delegatee, while still being able to accuse the delegatee in case of a dispute.[4]

Requirement 4. *One can not publicly verify whether a delegator or a delegatee created a certain report, while it is still possible to provide a proof assigning the task execution to one of the aforementioned parties.*

In addition, it is desirable to *automatically derive information*, e.g., to issue warnings if certain constraints in a workflow are not met.

Requirement 5. *Task reports allow to derive the order of the tasks in a certain workflow instance.*

3.1 Bringing Signatures to Workflows

We can model workflows using the well-known *delegation-by-certificate* approach from proxy signatures [37]. Subsequently, we describe two useful patterns.

Static assignment. Figure 2 illustrates the pattern for a statically assigned workflow. Here, the workflow manager computes a signature σ_0 on a sequence of (sets of) public keys PK together with the respective split/join operations. Then, for each task, (one of) the authorized agent(s) can sign the respective report using its secret key sk_i corresponding to the public key pk_i in PK. To be able to reconstruct the order of the task executions, agent A_j also includes the signature(s) of the agent(s) executing the preceding tasks in its signature σ_i.

Dynamic assignment. Figure 3 describes the pattern for dynamically assigned workflows. In this approach, the workflow manager only delegates to the first agent within the workflow and the agents can further delegate the execution rights for subsequent tasks to subsequent agents.

3.2 Structuring Task Reports

Orthogonal to the requirement to ensure logical relations among tasks, it might also be interesting to automatically verify certain constraints regarding particular decisions upon execution of a task t_i. For instance, it would be convenient to predefine certain sets of possible actions of an agent per task. As a simple realization one can think of a form containing several multiple-choice fields, where each multiple-choice field corresponds to a subtask of a specific task in a workflow. Then, an application monitoring reports can easily define constraints in

[4] When following the paradigm in Figure 2, the workflow manager is the delegator, whereas the agents are the delegatees. In contrast, following the paradigm in Figure 3, agents act as both, delegatees and delegators, while only the first delegation is performed by the workflow manager.

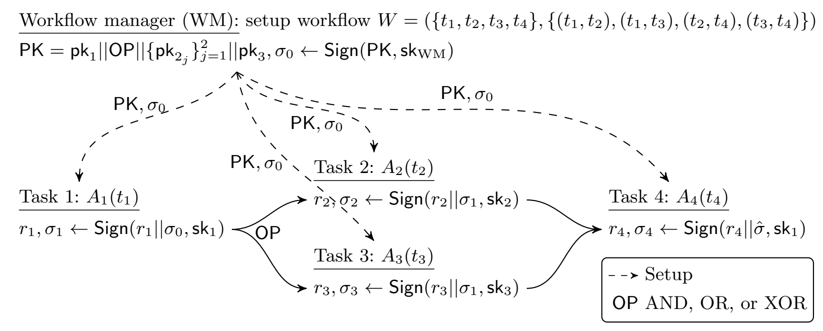
Workflow manager (WM): setup workflow $W = (\{t_1, t_2, t_3, t_4\}, \{(t_1, t_2), (t_1, t_3), (t_2, t_4), (t_3, t_4)\})$
$\mathsf{PK} = \mathsf{pk}_1 || \mathsf{OP} || \{\mathsf{pk}_{2_j}\}_{j=1}^{2} || \mathsf{pk}_3, \sigma_0 \leftarrow \mathsf{Sign}(\mathsf{PK}, \mathsf{sk}_{\mathrm{WM}})$
PK, σ_0
PK, σ_0
PK, σ_0
PK, σ_0
Task 1: $A_1(t_1)$
$r_1, \sigma_1 \leftarrow \mathsf{Sign}(r_1 || \sigma_0, \mathsf{sk}_1)$
OP
Task 2: $A_2(t_2)$
$r_2, \sigma_2 \leftarrow \mathsf{Sign}(r_2 || \sigma_1, \mathsf{sk}_2)$
Task 3: $A_3(t_3)$
$r_3, \sigma_3 \leftarrow \mathsf{Sign}(r_3 || \sigma_1, \mathsf{sk}_3)$
Task 4: $A_4(t_4)$
$r_4, \sigma_4 \leftarrow \mathsf{Sign}(r_4 || \hat{\sigma}, \mathsf{sk}_1)$
Setup
OP AND, OR, or XOR

Fig. 2. Pattern for statically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. If $\mathsf{OP} = \mathrm{AND}$ then $\hat{\sigma} \leftarrow \sigma_2 || \sigma_3$, if $\mathsf{OP} = \mathrm{OR}$ then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow \sigma_2 || \sigma_3$, if $\mathsf{OP} = \mathrm{XOR}$ then $\hat{\sigma} \leftarrow \sigma_2$ or $\hat{\sigma} \leftarrow \sigma_3$.

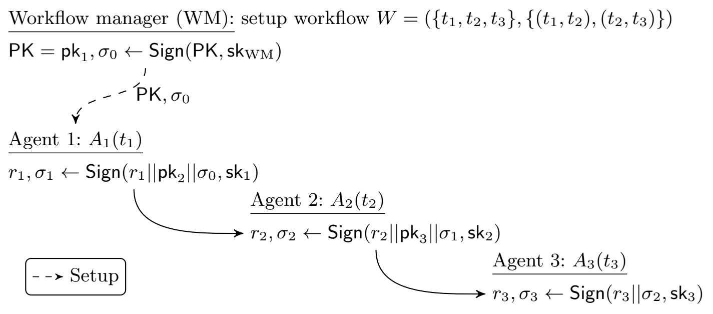
Workflow manager (WM): setup workflow $W = (\{t_1, t_2, t_3\}, \{(t_1, t_2), (t_2, t_3)\})$
$\mathsf{PK} = \mathsf{pk}_1, \sigma_0 \leftarrow \mathsf{Sign}(\mathsf{PK}, \mathsf{sk}_{\mathrm{WM}})$
PK, σ_0
Agent 1: $A_1(t_1)$
$r_1, \sigma_1 \leftarrow \mathsf{Sign}(r_1 || \mathsf{pk}_2 || \sigma_0, \mathsf{sk}_1)$
Agent 2: $A_2(t_2)$
$r_2, \sigma_2 \leftarrow \mathsf{Sign}(r_2 || \mathsf{pk}_3 || \sigma_1, \mathsf{sk}_2)$
Agent 3: $A_3(t_3)$
$r_3, \sigma_3 \leftarrow \mathsf{Sign}(r_3 || \sigma_2, \mathsf{sk}_3)$
Setup

Fig. 3. Pattern for dynamically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. For simplicity, we omit split/join (cf. Figure 2).

the fashion of: **if** *Option A was chosen in Subtask 1.1* **and** *Option B was chosen in Subtask 1.2* **then** *issue a warning.* If required, this can easily be extended to arbitrarily complex forms per task.

Adding a structure to the task reports, suggests to introduce the following additional requirements.

Requirement 6. *It is possible to predefine the structure of task reports.*

Besides addressing the structure of the report, allowing the delegatee to predefine sets of admissible choices for certain parts of task reports would help to improve the quality and help to automate the processing.

Requirement 7. *It is possible to predefine sets of admissible choices for certain fields in the task report.*

However, such detailed workflow reports also impose privacy requirements, since it is crucial that business internals remain confidential, e.g., when reports are revealed for auditing purposes.

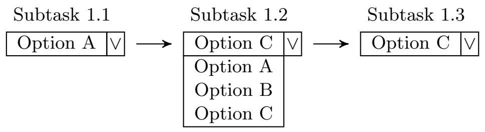

Fig. 4. A simple task report for a task $t_1 = (t_{1.1}, t_{1.2}, t_{1.3})$, composed of three multiple choice elements.

Requirement 8. *Task reports do not reveal additional information that is available to the delegator and/or the execution agent (e.g., the unused choices of the predefined sets of admissible replacements).*

4 Instantiations

Using standard digital signatures, one can straightforwardly instantiate the patterns in Figure 2 and Figure 3. Subsequently, we revisit the instantiation of these patterns with other variants of digital signatures. We stress that we provide algorithmic descriptions for the schemes as we believe that this makes the presentation unambiguous and clearer than any informal textual description.

Append-only Signatures. Append only signatures [32] allow to publicly extend signed messages and to update the signature correspondingly. An append only signature scheme (AOS) is a tuple of efficient algorithms (Setup, Append, Verify), which are defined as follows:

Setup : On input of a security parameter κ, this algorithm outputs a keypair $(\mathsf{sk}, \mathsf{pk})$, where sk constitutes the signature on the empty message.

Append : On input of a public key pk, a signature σ_{n-1} on a message $(m_1, \ldots, m_{n-1})$, and a message m_n, this algorithm outputs a signature σ_n on the message $(m_1, \ldots, m_n)$.

Verify : On input of a public key pk, a signature σ and a message $M = (m_1, \ldots, m_n)$, this algorithm outputs a bit $b \in \{0, 1\}$, indicating whether σ is valid.

For security, AOS are required to provide AOS-unforgeability under chosen message attacks. Informally this means that the only way of creating a valid signature of length n on a message $M = (m_1, \ldots, m_n)$ is to extend a valid signature on message $M' = (m_1, \ldots, m_{n-1})$.

Application to Workflows: Using append-only signatures, the workflow manager creates a signature on the empty message and each agent can append its documentation. Due to their public-append capabilities, AOS are suited for unauthorized delegations, which only ensure the integrity of the signed reports.

Redactable Signatures. Informally, redactable signatures [31,38,48] allow to sign documents, where certain predefined parts can later be blacked out (or cloaked) without signer interaction and without invalidating the signature. A redactable signature scheme (RSS) is a tuple of efficient algorithms (KeyGen, Sign, Verify, Redact), which are defined as follows (using the notation of [19]):

KeyGen : On input of a security parameter κ, this algorithm outputs a key-pair (sk, pk).

Sign : On input of a secret key sk, a message M and admissible redactions ADM, this algorithm returns a message-signature pair (M, σ) (where ADM can be derived from σ).

Verify : On input of a public key pk, a message M and a signature σ, this algorithm outputs a bit $b \in \{0, 1\}$, indicating the validity of σ.

Redact : This algorithm takes a public key pk, a signature σ, a message M and modification instructions MOD, computes an updated signature σ' and outputs an updated message signature pair $(\mathsf{MOD}(M), \sigma')$.

Essentially the redaction can be done by everyone, meaning that (1) the entity that performs the redaction is not accountable for the changes and (2) one is only able to *black out* certain document parts. For security, redactable signatures are required to be *unforgeable* and *private*.

Unforgeability captures the infeasibility to output a valid message signature pair (M, σ) without knowing sk, unless (M, σ) was obtained by redaction.

Privacy requires it to be infeasible for every efficient adversary to reconstruct the redacted message parts, given the redacted message and its signature.

See [19] for a formal security model. Besides these properties, the security model for RSS has been refined and extended several times. Firstly, [45] introduced the notion of *accountability*, which requires that signers and redactors can be held accountable for their signatures/redactions. Secondly, [45] and [14] independently introduced *unlinkability* for RSS as an even stronger privacy notion. *Unlinkability* essentially requires multiple redactions of the same document to be unlinkable. We, however, note that we do not further consider unlinkability here, since privacy already provides the required security guarantees in our context. We also mention that redactable signatures are related to the more general framework of P-homomorphic signatures [1].

Application to Workflows: In context of workflows, RSS can be used in two different ways:

(1) One uses RSS in the same way as conventional DSS. Then, when it is required to publish reports (e.g., for auditing purposes) it can be useful to redact certain confidential parts of the reports.

(2) Provided that all potential reports are known prior to designating a task to an agent, one could enumerate all variants of the reports and sign this list using an RSS. The agent then simply redacts—thus removes—all reports that are not required. While conventional RSS do not provide accountability in this setting, accountable RSS (ARSS) [45] can be used to additionally provide accountability.

Sanitizable Signatures. Sanitizable signatures [3,9,10,11,13,12,46] split messages in fixed and variable message parts and allow to issue signatures on them. A designated party (the sanitizer) is then able to modify the variable parts of the message without invalidating the signature. A sanitizable signature scheme (SSS) is a tuple of efficient algorithms ($\mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge}$). Subsequently, we recall the definitions from [9]:

$\mathsf{KeyGen}_{\mathsf{sig}}$: On input of a security parameter κ, this algorithm outputs a signer key-pair $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$.

$\mathsf{KeyGen}_{\mathsf{san}}$: On input of a security parameter κ, this algorithm outputs a sanitizer key-pair $(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}})$.

Sign : On input of a message M, corresponding admissible modifications ADM, the keypair of the signer $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$, as well as the verification key of the sanitizer $\mathsf{pk}_{\mathsf{san}}$, this algorithm outputs a message-signature pair (M, σ), where it is assumed that ADM can be reconstructed from σ.

Sanit : On input of a valid message-signature pair (M, σ), modification instructions MOD, some auxiliary information aux, the verification key of the signer $\mathsf{pk}_{\mathsf{sig}}$ and the secret key of the sanitizer $\mathsf{sk}_{\mathsf{san}}$, this algorithm outputs an updated message signature pair $(\mathsf{MOD}(m), \sigma')$ and $\perp$ if the modification instructions are incompatible with ADM.

Verify : On input of a message-signature pair (M, σ) and the verification keys of the signer $\mathsf{pk}_{\mathsf{sig}}$ and the sanitizer $\mathsf{pk}_{\mathsf{san}}$, this algorithm outputs a bit $b \in \{0, 1\}$ indicating whether σ is a valid signature on M.

Proof : On input of a message-signature pair (M, σ), q message-signature pairs $(M_j, \sigma_j)_{j=1}^{q}$ created by the signer, the keypair $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$ of the signer and the verification key of the sanitizer $\mathsf{pk}_{\mathsf{san}}$, this algorithm outputs a proof π.

Judge : On input of a message-signature pair (M, σ), the verification keys of the signer $\mathsf{pk}_{\mathsf{sig}}$ and the sanitizer $\mathsf{pk}_{\mathsf{san}}$, and a valid a proof π, this algorithm outputs a bit $b \in \{\mathtt{sig}, \mathtt{san}\}$, indicating whether the respective signature was created by the signer or the sanitizer.

Subsequently, we informally discuss the security properties of sanitizable signatures (introduced in [3] and formalized in [9]):

Unforgeability requires that only honest signers and sanitizers are able to produce valid signatures.

Immutability requires that malicious sanitizers are not able to modify fixed message parts.

Transparency requires that no one (except the signer and the sanitizer) can distinguish signatures of the signer from signatures of the sanitizer.

Privacy requires that no one (except the signer and the sanitizer) can recover sanitized information.

Signer-/Sanitizer-accountability Requires that no signer can falsely accuse a sanitizer of having created a certain signature and vice versa.

The above properties have seen some refinement and gradual extension since their formalization in [9], e.g., by [33,25,16,13,12,45,20].

In [33], among others, an extension that additionally allows to define sets of admissible replacements per message block (`LimitSet`) was introduced and later formalized in [15] (henceforth called extended sanitizable signatures or ESSS). Their formalization, however, does not require the sets of admissible modifications to remain concealed upon verification, and, thus, does not define privacy in the original sense. Thus, [20] introduced the notion of *strong privacy*, that additionally covers this requirement. In [20], it is also shown that ESSS providing strong privacy can be black-box constructed from every secure SSS in the model of [9] and indistinguishable accumulators [18].

Orthogonal to that, [13] discusses that accountability can be modeled in two ways: non-interactive or interactive. The model presented above is tailored to interactive (non-public) accountability. In contrast, non-interactive (public) accountability requires that Judge works correctly on an empty proof π.[5] We emphasize that non-interactive accountability might be helpful in workflows, where the original signer can not be involved for certain reasons, e.g., efficiency.

Application to Workflows: By definition, SSS include a delegation mechanism, i.e., a signer grants a sanitizer permission to modify certain parts of a signed message without invalidating the signature. Thus, using this primitive, one can not only pre-specify the execution agent, but also the structure of the report.[6] In other words, SSS allow to split the report into several fields; then, according to the pre-defined workflow, one specifies which agent (i.e., by specifying the sanitizer) is allowed to put arbitrary content into certain fields of the report. In addition, SSS provide *transparency*, which is useful if it is required to hide whether a certain task was outsourced or not. In case of a dispute, the {Proof, Judge} algorithms still guarantee accountability. In case the additional level of privacy given by transparency is not needed, one can use non-interactively (publicly) accountable SSS, e.g., [13,12].

ESSS [15]: Extended sanitizable signatures, as defined in [15], extend SSS by the possibility to limit the admissible modifications per message block to sets of allowed messages. This allows for an even more fine grained definition of the report structure. However, the model of [15] does not require the unused choices in the sets of admissible modifications to remain hidden upon verification. While this extension eases the automatic processing of reports, the limited privacy features limit the practical applicability of this instantiation.

ESSS [20]: Extended sanitizable signatures, as defined in [20], fix the aforementioned privacy problems, which, in turn, extends their applicability to workflow documentation systems.

Proxy Signatures. Proxy signature (PS) schemes, introduced in [37] and formalized in [7] allow a delegator to delegate the signing rights for a certain message

[5] Note that this obviously contradicts transparency, meaning that no scheme can be transparent and non-interactively accountable at the same time.

[6] Using SSS supporting multiple sanitizers [16], one can even pre-specify multiple possible agents for a single task.

space $\mathcal{M}$ to a proxy. A proxy can then produce signatures for messages $m \in \mathcal{M}$ on behalf of the delegator. Subsequently, we recall the definitions from [7]:

(D, P) : The originator and the proxy jointly compute a delegation for the message space $\mathcal{M}$ as well as a proxy signing key skp. The originator runs D and outputs the delegation σ computed using its signing key sk_i, whereas the proxy verifies the delegation and obtains the proxy signing key skp, which consists of its private signing key sk_j and the originators delegation.

Sign : This algorithm computes and outputs a proxy signature σ_P for message $m \in \mathcal{M}$ using the proxy signing key skp.

Verify : This algorithm verifies whether proxy signature σ_P is a valid proxy signature for message m under pk_j, delegated by pk_i. On success, this algorithm outputs 1, and 0 otherwise.

ID : This algorithm outputs the identity j of the proxy, when given a proxy signature σ_P.

For security, proxy signatures are required to be *unforgeable*, which informally means that no one can produce valid signatures for messages $m \notin \mathcal{M}$ and only the designated proxy can produce valid signatures for $m \in \mathcal{M}$. In [27], the model was extended by introducing *privacy*, which essentially requires that—upon verification of a signature σ_P on a message $m \in \mathcal{M}$—the verifier learns nothing about $\mathcal{M}$ (except that $m \in \mathcal{M}$). Signatures secure in this model are called warrant-hiding proxy signatures (WHPS). We note that proxy signatures are one instantiation of the more general concept of functional signatures [5,8].

Application to Workflows: Here, a delegator grants a proxy the signing rights for messages out of a certain message space $\mathcal{M}$. This delegation mechanism can be used to predefine all possible reports and the executing agent only chooses the suitable report. In addition, WHPS additionally provide privacy with respect to the unused reports in the designated message space.

Blank Digital Signatures. Blank digital signatures, introduced in [26], allow an originator O to define and sign forms (so-called templates $\mathcal{T}$) consisting of fixed and exchangeable (multiple-choice) elements. These forms can then be filled in (instantiated) by a designated party (the proxy P). Upon verification, the verifier only learns the values chosen by the designated party. A blank digital signature scheme (BDSS) is a tuple of efficient algorithms ($\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}_\mathsf{T}, \mathsf{Inst}, \mathsf{Verify}_\mathsf{I}$), which are introduced subsequently. Thereby, we assume that DSS signing keys for the originator $(\mathsf{sk}_\mathsf{O}, \mathsf{pk}_\mathsf{O})$ and the proxy $(\mathsf{sk}_\mathsf{P}, \mathsf{pk}_\mathsf{P})$ already exist.

KeyGen : On input of a security parameter κ and an upper bound for the template size t, this algorithm outputs public parameters pp. We assume pp to be an implicit input to all subsequent algorithms.

Sign : On input of a template $\mathcal{T}$, the signing key of the originator sk_O and the verification key of the proxy pk_P, this algorithm outputs a template signature $\sigma_\mathcal{T}$ and a secret instantiation key $\mathsf{sk}_\mathsf{P}^\mathcal{T}$ for the proxy.

$\mathsf{Verify_T}$: On input of a template $\mathcal{T}$, a template signature $\sigma_{\mathcal{T}}$, the instantiation key of the proxy $\mathsf{sk}_{\mathsf{P}}^{\mathcal{T}}$ and the public verification keys of the originator $\mathsf{pk_O}$ and the proxy $\mathsf{pk_P}$, this algorithm outputs a bit $b \in \{0, 1\}$, indicating whether $\sigma_{\mathcal{T}}$ is valid.

Inst : On input of a template $\mathcal{T}$, a template signature $\sigma_{\mathcal{T}}$, an instance $\mathcal{M}$, the signing key $\mathsf{sk_P}$ and the instantiation key $\mathsf{sk}_{\mathsf{P}}^{\mathcal{T}}$ of the proxy, this algorithm outputs an instance signature $\sigma_{\mathcal{M}}$ on $\mathcal{M}$ if $\mathcal{M}$ is a valid instance of $\mathcal{T}$ and $\perp$ otherwise.

$\mathsf{Verify_I}$: On input of an instance $\mathcal{M}$, an instance signature $\sigma_{\mathcal{M}}$ and the verification keys of the originator $\mathsf{pk_O}$ and the proxy $\mathsf{pk_P}$, this algorithm outputs a bit $b \in \{0, 1\}$, indicating whether $\sigma_{\mathcal{M}}$ is valid.

The security requirements for BDSS are (informally) defined as follows:

Unforgeability requires that only the honest originator and proxy can create valid signatures.

Immutability requires that even malicious proxies cannot create instance signatures for invalid instances $\mathcal{M}$ of $\mathcal{T}$.

Privacy requires that no one (except the proxy and the originator) can recover the unused choices for the exchangeable elements.

Application to Workflows: BDSS are—up to the missing transparency and accountability properties—similar to ESSS in [20] and can, thus, be used for similar purposes. We note that all known instantiations of BDSS [26,17] provide public accountability, since they require an explicit signature of the delegatee (proxy).

4.1 Comparison and Discussion

In Table 1, we bring the various possible instantiations discussed above into the context of the previously defined requirements, where we exclude naive instantiations. Depending on the used scheme, we can cover different subsets of previously

Inst.	R1	R2	R3	R4	R5	R6	R7	R8
DSS	✓	✓	✓		✓			
AOS	✓				✓			
RSS (1)	✓	✓	✓		✓			✓
RSS (2)	✓	✓	A	A†	✓			✓
SSS	✓	✓	✓	✓†	✓	✓		
ESSS [15]	✓	✓	✓	✓†	✓	✓	✓	
ESSS [20]	✓	✓	✓	✓†	✓	✓	✓	✓
PS	✓	✓	✓		✓			
WHPS	✓	✓	✓		✓			✓
BDSS	✓	✓	✓		✓	✓	✓	✓

Table 1. Requirements covered by the respective instantiations. Legend: ✓ . . . supported, A . . . supported by ARSS [45], † . . . if scheme is transparent

posed requirements. While choosing a concrete instantiation always depends on the requirements, we note that ESSS and BDSS seem to be particularly well suited for the considered applications. Note that—mainly due to the imposed overhead—we do not consider naive solutions such as achieving Requirement 6 and 7 by enumerating all possible task reports. We again stress that the instantiations discussed in this paper are very simple and only make use of standard cryptographic primitives with multiple efficient instantiations. Thereby, we only require to assume the existence of some public key authority.

Outlook. In this paper we have discussed potential solutions for authentic and accountable, yet privacy maintaining documentation of outsourced workflows. While we, thereby, followed a rather high-level and informal approach, it would be interesting to model the desired security properties more formally (as for instance done in [36] for cloud provenance). Furthermore, it would be interesting to evaluate the practical value of our proposed solutions in a real world setting. Finally, we note that it seems to be straight forward to extend our approach by cryptographic access control solutions (e.g., [21,22]) to restrict the access to task reports. We leave these points as future work.

References

1. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. J. Cryptology 28(2) (2015)
2. Al-Riyami, S.S., Malone-Lee, J., Smart, N.P.: Escrow-free encryption supporting cryptographic workflow. Int. J. Inf. Sec. 5(4) (2006)
3. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable signatures. In: ESORICS 2005. LNCS, vol. 3679
4. Bagga, W., Molva, R.: Policy-based cryptography and applications. In: FC 2005
5. Bellare, M., Fuchsbauer, G.: Policy-Based Signatures. In: PKC 2014. LNCS, vol. 8383
6. Bier, C.: How usage control and provenance tracking get together - a data protection perspective. In: IEEE Security and Privacy Workshops (SPW) 2013. IEEE
7. Boldyreva, A., Palacio, A., Warinschi, B.: Secure proxy signature schemes for delegation of signing rights. J. Cryptology 25(1) (2012)
8. Boyle, E., Goldwasser, S., Ivan, I.: Functional Signatures and Pseudorandom Functions. In: PKC 2014. LNCS, vol. 8383
9. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of sanitizable signatures revisited. In: PKC 2009. LNCS, vol. 5443
10. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Sanitizable signatures: How to partially delegate control for authenticated data. In: BIOSIG 2009. LNI, vol. 155
11. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of sanitizable signatures. In: PKC 2010. LNCS, vol. 6056
12. Brzuska, C., Pöhls, H.C., Samelin, K.: Efficient and perfectly unlinkable sanitizable signatures without group signatures. In: EuroPKI 2013. LNCS, vol. 8341
13. Brzuska, C., Pöhls, H.C., Samelin, K.: Non-interactive public accountability for sanitizable signatures. In: EuroPKI 2012. LNCS, vol. 7868

14. Camenisch, J., Dubovitskaya, M., Haralambiev, K., Kohlweiss, M.: Composable & modular anonymous credentials: Definitions and practical constructions. In: ASIACRYPT 2015. LNCS, vol. 9453
15. Canard, S., Jambert, A.: On extended sanitizable signature schemes. In: CT-RSA 2010. LNCS, vol. 5985
16. Canard, S., Jambert, A., Lescuyer, R.: Sanitizable signatures with several signers and sanitizers. In: AFRICACRYPT 2012. LNCS, vol. 7374
17. Derler, D., Hanser, C., Slamanig, D.: Privacy-enhancing proxy signatures from non-interactive anonymous credentials. In: DBSec 2014. LNCS, vol. 8566
18. Derler, D., Hanser, C., Slamanig, D.: Revisiting cryptographic accumulators, additional properties and relations to other primitives. In: CT-RSA 2015. LNCS, vol. 9048
19. Derler, D., Pöhls, H., Samelin, K., Slamanig, D.: A general framework for redactable signatures and new constructions. In: ICISC 2015. LNCS, vol. 9558
20. Derler, D., Slamanig, D.: Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In: ProvSec 2015. LNCS, vol. 9451
21. Ferrara, A.L., Fuchsbauer, G., Liu, B., Warinschi, B.: Policy privacy in cryptographic access control. In: CSF 2015. IEEE
22. Ferrara, A.L., Fuchsbauer, G., Warinschi, B.: Cryptographically enforced RBAC. In: CSF 2013. IEEE
23. Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for computational tasks: A survey. Computing in Science & Engineering 10(3) (2008)
24. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 17(2) (1988)
25. Gong, J., Qian, H., Zhou, Y.: Fully-secure and practical sanitizable signatures. In: InsCrypt 2010. LNCS, vol. 6584
26. Hanser, C., Slamanig, D.: Blank digital signatures. In: AsiaCCS 2013. ACM
27. Hanser, C., Slamanig, D.: Warrant-hiding delegation-by-certificate proxy signature schemes. In: INDOCRYPT 2013. LNCS, vol. 8250
28. Herkenhöner, R., Jensen, M., Pöhls, H.C., de Meer, H.: Towards automated processing of the right of access in inter-organizational web service compositions. In: WSBPS 2010. IEEE
29. ISO/IEC 19510: Information technology – Object Management Group Business Process Model and Notation (2013)
30. Jablonski, S.: On the complementarity of workflow management and business process modeling. SIGOIS Bull. 16(1) (1995)
31. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic signature schemes. In: CT-RSA 2002. LNCS, vol. 2271
32. Kiltz, E., Mityagin, A., Panjwani, S., Raghavan, B.: Append-only signatures. In: ICALP 2005. LNCS, vol. 3580
33. Klonowski, M., Lauks, A.: Extended sanitizable signatures. In: ICISC 2006. LNCS, vol. 4296
34. Lim, H.W., Kerschbaum, F., Wang, H.: Workflow signatures for business process compliance. IEEE Trans. Dependable Sec. Comput. 9(5) (2012)
35. Lim, H.W., Paterson, K.G.: Multi-key hierarchical identity-based signatures. In: IMACC 2007. LNCS, vol. 4887
36. Lu, R., Lin, X., Liang, X., Shen, X.S.: Secure provenance: the essential of bread and butter of data forensics in cloud computing. In: ASIACCS 2010. ACM
37. Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures for delegating signing operation. In: CCS 1996. ACM

38. Miyazaki, K., Iwamura, M., Matsumoto, T., Sasaki, R., Yoshiura, H., Tezuka, S., Imai, H.: Digitally signed document sanitizing scheme with disclosure condition control. IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences 88-A(1) (2005)
39. Montagut, F., Molva, R.: Enforcing integrity of execution in distributed workflow management systems. In: SCC 2007. IEEE
40. Montagut, F., Molva, R.: Traceability and integrity of execution in distributed workflow management systems. In: ESORICS 2007. LNCS, vol. 4734
41. Moreau, L., Groth, P., Miles, S., Vazquez-Salceda, J., Ibbotson, J., Jiang, S., Munroe, S., Rana, O., Schreiber, A., Tan, V., et al.: The provenance of electronic data. Communications of the ACM 51(4) (2008)
42. Moreau, L., Ludäscher, B., Altintas, I., Barga, R.S., Bowers, S., Callahan, S., Chin, G., Clifford, B., Cohen, S., Cohen-Boulakia, S., et al.: Special issue: the first provenance challenge. Concurrency and Computation: Practice and Experience 20(5) (2008)
43. Paterson, K.: Cryptography from pairings: A snapshot of current research. Information Security Technical Report 7(3) (2002)
44. Pearson, S., Tountopoulos, V., Catteddu, D., Südholt, M., Molva, R., Reich, C., Fischer-Hübner, S., Millard, C., Lotz, V., Jaatun, M.G., Leenes, R., Rong, C., Lopez, J.: Accountability for cloud and other future internet services. In: CloudCom 2012. IEEE
45. Pöhls, H.C., Samelin, K.: Accountable redactable signatures. In: ARES 2015. IEEE
46. Pöhls, H.C., Samelin, K., Posegga, J.: Sanitizable signatures in XML signature - performance, mixing properties, and revisiting the property of transparency. In: ACNS 2011. LNCS, vol. 6715
47. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. ACM Sigmod Record 34(3) (2005)
48. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: ICISC 2001. LNCS, vol. 2288