prisma cloud

Analysis of Malleable Signatures for Defining Allowed Modification and Providing Verifiable Means of Conformant Processing

(The Final Design of the FLEXAUTH Tool)

Deliverable D5.9

Editor Name	Henrich C. Pöhls (UNI PASSAU)
Type	Report
Dissem. Level	PU
Release Date	31.05.2017
Version	1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644962.

More information available at https://prismacloud.eu.

Copyright Statement

Executive Summary

PRISMACLOUD aims at bringing novel cryptographic concepts and methods to practical application to improve the security and privacy of cloud based services and make them usable for providers and users.

The purpose of this deliverable is to present the final version of the FLEXAUTH tool including the architecture and the design of the tool, starting from terms and definitions, over design paradigms and component model, to the static software architecture. This tool provides means for flexible authentication protocols with selective disclosure and is based on variants of malleable signature schemes , i.e., redactable signature schemes (RSS) as well as group signature schemes (GSS). When verifying a signature with one of the cryptographic schemes available through the FLEXAUTH tool, one verifies (directly or indirectly) whether a certain piece of data is conformant to a certain policy. The special-purpose signatures inside FLEXAUTH thus allow to verify the conformance with different types of policies with different expressiveness. While the aforementioned aspects represent a final iteration of content already present in Deliverable D5.6., we additionally present research regarding the application of the FLEXAUTH tool beyond the application in the e-Health and Smart City use-cases within PRISMACLOUD.

Table of Contents

Executive Summary **1**

1 Introduction **6**
1.1 Scope of the Document . . . 7
1.2 Relation to Other Project Work . . . 7
1.3 Structure of the Document . . . 8

2 The FLEXAUTH Tool **9**
2.1 Overview . . . 9
2.2 Tool Architecture . . . 9
2.3 Component Model . . . 9
2.4 Software Implementation . . . 11
2.5 Services Based on FLEXAUTH . . . 11
2.6 Terms and Definitions . . . 12
2.6.1 Signer . . . 12
2.6.2 Redactor/Sanitizer . . . 12
2.6.3 Issuer . . . 12
2.6.4 Opener . . . 12
2.6.5 Linker . . . 12
2.6.6 Verifier . . . 12
2.6.7 Message . . . 13
2.6.8 Accumulator . . . 13
2.6.9 Group . . . 13
2.6.10 Redaction . . . 13
2.6.11 Sanitization . . . 13
2.6.12 Signature Scheme . . . 13
2.6.13 Signature . . . 13
2.6.14 Group Signature . . . 14
2.6.15 Selective Disclosure Token . . . 14
2.6.16 Redacted Signature . . . 14
2.6.17 Sanitized Signature . . . 14
2.6.18 Group Signature Scheme . . . 14
2.6.19 Redactable Signature Scheme . . . 14
2.6.20 Sanitizable Signature Scheme . . . 14

2.6.21 Admissible Changes . . . 14
2.6.22 Modification Instructions . . . 15
2.6.23 Redacted Message . . . 15
2.6.24 Sanitized Message . . . 15
2.6.25 Verification key . . . 15
2.6.26 Signing key . . . 15
2.6.27 Sanitization key . . . 15
2.6.28 Issuing Key . . . 16
2.6.29 Opening Key . . . 16
2.6.30 Linking Key . . . 16
2.6.31 Controllable Linkability . . . 16
2.7 Malleable Signatures Library . . . 16
2.7.1 Abstract Description . . . 16
2.7.2 Architecture and Design of the Library . . . 20
2.8 Group Signatures Library . . . 23
2.8.1 Abstract Description . . . 23
2.8.2 Architecture and Design of the Library . . . 24
2.9 Recommendations . . . 25

3 The FLEXAUTH Tool in the Application Context 27
3.1 The eHealth Pilot . . . 27
3.2 The Smart City Pilot . . . 27
3.3 Research on Additional Applications . . . 28
3.3.1 Cryptographically Enforced Four-Eyes Principle . . . 28
3.3.2 Accountable and Privacy Preserving Workflows . . . 28
3.4 Extending Redactable Signatures with Additional Privacy Features . . . 29

4 Conclusion 30

List of Acronyms 31

List of Figures 31

Bibliography 34

A Appendix 35

Document information

Project Context

Work Package	WP5 Efficient and Secure Implementations
Task	T5.3 Secure and privacy preserving processing of authenticated data
Dependencies	D4.4, D4.5, D4.6, D4.7, D5.6, D6.5, D6.6, D7.8

Author List

Organization	Name	E-mail
TU Graz	Daniel Slamanig	daniel.slamanig@tugraz.at
TU Graz	David Derler	david.derler@iaik.tugraz.at
UNI PASSAU	Marek Klein	km@sec.uni-passau.de
UNI PASSAU	Henrich C. Pöhls	hp@sec.uni-passau.de
UNI PASSAU	Christoph Frädrich	fraedric@fim.uni-passau.de
UNI PASSAU	Paul Hoguth	prismacloud-passau@sec.uni-passau.de
UNI PASSAU	Christoph Hanschke	prismacloud-passau@sec.uni-passau.de

Reviewer List

Organization	Name	E-mail
TUDA	Lucas Schabhüser	lschabhueser@cdc.informatik.tu-darmstadt.de
XiTRUST	Katrin Riemer	Katrin.Riemer@xitrust.com

Version History

Version	Date	Reason/Change	Editor
0.1	2017-02-15	1st Draft	Henrich C. Pöhls
0.2	2017-02-17	Added two joint results, which have been published	Henrich C. Pöhls
0.3	2017-02-20	ToC draft	Henrich C. Pöhls
0.4	2017-04-20	Updated content and ToC before meeting	Henrich C. Pöhls
0.5	2017-04-23	Finalised ToC and some content	Pöhls et al.
0.6	2017-05-25	Architecture updates; Applications & Research items	D. Derler, D. Slamanig
0.7	2017-05-29	Version for Review	all
1.0	2017-05-31	Incorporating Final Changes & Comments	all

1 Introduction

The major purpose of this document is to describe the final iteration of the FLEXAUTH tool. Following the PRISMACLOUD architecture, FLEXAUTH is a tool (see Fig. 1) which offers the software implementations of a set of special-purpose signature schemes. In line with PRISMACLOUD's vision of a tool, FLEXAUTH bundles certain cryptographic primitives and abstracts away their complexity by offering high level and easy to use application programming interfaces (APIs). The architecture of FLEXAUTH is inspired by the architecture of the Java cryptography extensions (JCE), which (1) ensures easy extensibility in the sense that it is easy to extend the API to support other, related schemes, and (2) allows to straight-forwardly switch between different implementations of certain primitives with very little changes in the code which uses FLEXAUTH.

The FLEXAUTH API currently offers high-level access to redactable signature schemes (RSS), which constitute a special form of malleable signature schemes (MSS), as well as group signature schemes (GSS). In the context of RSS, the FLEXAUTH API uses a generic approach to redactable signatures developed within WP 4 of PRISMACLOUD (cf. Deliverable D4.8). This generic approach to RSS is in line with our design philosophy, in that it allows to generically instantiate redactable signatures from multiple diverse lower level cryptographic building blocks. Our API reflects this design and allows to easily control which underlying primitives are used in a concrete instantiation. For example one can easily plug in code to generate signatures using a key which is stored in software, but one can also easily plug in alternative, more sophisticated, means to generate the signatures. In the context of GSS, the FLEXAUTH API follows the same principles and it is designed in line with the most common model for dynamic group signatures (BSZ) known and well accepted in the scientific community.

This report presents the final architecture and design for the set of software libraries that in combination define the FLEXAUTH tool. Details on the implementation are already presented in Deliverable D6.5 and a final iteration thereof will be presented in D6.6.

Within PRISMACLOUD the prime application of the FLEXAUTH tool is within the eHealth pilot as well as the Smart City pilot. Nevertheless, it needs to be stressed that the design of FLEXAUTH is flexible and of course the provided schemes have applications in various other domains. In particular, within PRISMACLOUD we conducted research on the applicability of this tool in other application scenarios and we present the results in this deliverable.

Relation Between Title And Subtitle. Essentially, FLEXAUTH allows to verify (directly or indirectly) whether a certain piece of data is conformant to a certain policy. FLEXAUTH provides means to generate special-purpose signatures which allow to verify the conformance with different types of policies with different expressiveness. The semantics of such a policy thereby may vary from "only allowed modifications were applied to a certain piece of data" as it is the case for MSS, to "the signed data stems from some member of a group" as it is the case for GSS.

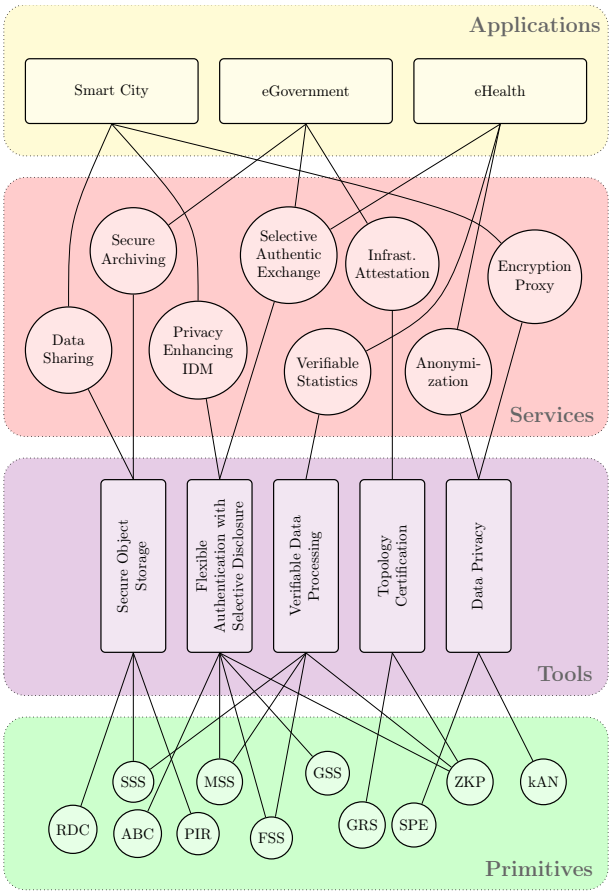

Figure 1: PRISMACLOUD architecture.

1.1 Scope of the Document

This deliverable is an iteration of Deliverable D5.6 and represents the final specification of the FLEXAUTH tool offering flexible authentication with selective disclosure functionality. Deliverable D5.6 already provided an outline of the architecture and a design of the tools, starting from terms and definitions, over design paradigms and component model, to the static software architecture. Compared to D5.6, this deliverable augments the software architecture of the part of the tool that realizes GSS and in addition provides research results investigating the use of the FLEXAUTH tool beyond the application within the PRISMACLOUD pilots.

1.2 Relation to Other Project Work

This deliverable relates to the WP4 and the corresponding research into cryptographic primitives in that it offers a concrete design for the realization of these primitives in software. This deliverable is thereby related to the deliverables D4.6 and D4.7, representing *First Year Research on Privacy-Enhancing Cryptography*, as well as the *Progress Report on*

Privacy-Enhancing Cryptography, respectively. Moreover, cryptographic techniques used in this report are also related to D4.4 *Overview of Functional and Malleable Signature Schemes*. Finally, with D4.5, the (also related) final report on *Signature Schemes Allowing for Verifiable Operations on Authenticated Data* is going to be delivered only two months after this deliverable. The design of the tool is closely interlinked with the cryptographic research done in WP4. This corresponds to PRISMACLOUD's development methodology based on the 4-tier architecture that calls for a close cooperation between cryptographers (WP4) and software developers (WP5 and WP6). It is embedded in PRISMACLOUD's Cryptographic Service Development LiveCycle (CryptSDLC) that was described first in D7.3 and shows how to blend security-by-design approaches with a workflow suitable for cryptographic design.

With the tool and its functionality being described, this deliverable further lays the foundations for upper layers of the PRISMACLOUD architecture that incorporate the functionality emitted by the tool (D7.4, D7.6, D7.8). In those deliverables the tools will be used to build new cloud services that can offer stand alone privacy and security enhanced functionality, or to build micro-services to be incorporated into existing cloud services to enhance their privacy features. Finally, we also want to note that the description of the terminology of group signatures and redactable signatures given in Section 2.6 is also very helpful for future communication between cryptographers and developers and will help PRISMACLOUD's WP9 effort to communicate with a standardized set of terms. It might even be used in PRISMACLOUD's strive to standardize redactable signature schemes (details on the standardization efforts are/will be presented in the WP9 deliverables).

1.3 Structure of the Document

This deliverable is structured into two main sections. The first major section (Section 2) constitutes a revision of Section 2 from D5.6 and delivers all the information on the final design and architecture of the FLEXAUTH tool. In this section, we first give a high-level overview of the tools structure and define terms and definitions which are important to set the stage for the description of the respective tools. Then, we describe the component model and the API design of the tools, including the underlying design paradigms. Finally, we conclude with recommendations to be adhered for a secure implementation of the tools. The second major section (Section 3) sets the FLEXAUTH tool into the context of the pilot use cases of PRISMACLOUD, and also presents our research on possible additional applications beyond the pilot use cases of PRISMACLOUD.

2 The FLEXAUTH Tool

On a high level, the flexible authentication with selective disclosure (FLEXAUTH) tool allows to authenticate arbitrary messages (documents) so that a (dedicated) party can selectively disclose partial information of the original message (document)—according to some (well) defined rules—while the original signature (from the originator) remains valid. In the most extreme case this just amounts to demonstrate that one is in possession of some authenticated message from some signer with enhanced privacy, i.e., the message is not disclosed and thus the disclosing party remains anonymous.

2.1 Overview

This tool heavily relies on *malleable signature schemes* (and in particular *redactable signatures* and *sanitizable signatures*), i.e., signature schemes that allow to modify already signed messages in a controlled way, without signer-interaction (access to the secret signing key), while preserving the validity of the original signature respectively. Additionally, it relies on the privacy-enhancing cryptographic primitive denoted as *group signature schemes*, which allows users to join a group (getting issued a so called membership certificate by some signer) and then be able to demonstrate possession of such a signature (which can be used to authenticate any other message) without revealing the identity, i.e., only membership in a certain group is revealed.

2.2 Tool Architecture

The architecture of the FLEXAUTH tool is kept very simplistic (cf. Figure 2). Its main functionality that is exposed to the PRISMACLOUD services is encapsulated in two libraries, i.e., the *Malleable Signatures* and the *Group Signatures Libraries*, both being entirely developed within PRISMACLOUD. The libraries itself can be extended with new algorithms that are written as providers. The best example is the the IAIK JCE provider. All the basic arithmetic and cryptographic functionality required by these two libraries is provided by the IAIK JCE provider and specific functionality for elliptic curve and pairing based cryptography is provided by its extension denoted ECCelerate^{TM}.

2.3 Component Model

The abstract components of the FLEXAUTH tool are illustrated in Figure 3. Subsequently, we briefly point to how they are going to be used in the two PRISMACLOUD services related to this tool (because the single components are used differently in the two services).

Authentication Component. Using the authentication component (Auth), an entity can obtain a verifiably authentic version of a message. This may either be a conventional

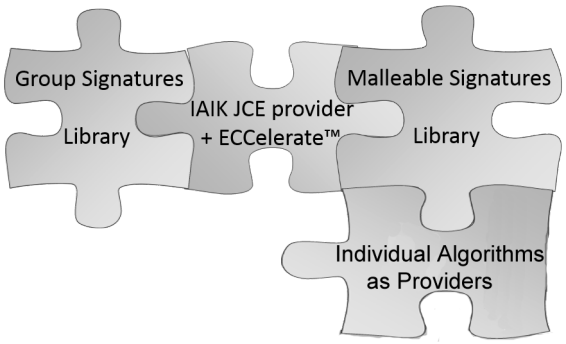

Figure 2: Library components of the FLEXAUTH tool.

signature on some message (document) or may be a (special) signature by some entity on some specific information (e.g., the certification of some group membership). Consequently, the authentication component operates on (semi-)structured documents and produces (malleable) signatures attesting the authenticity of the respective documents or represent the joining of some well defined group. Thus, the choice of signature schemes ranges from classical to redactable signatures (or sanitizable signatures) relying on any conventional digital signature scheme (such as DSA, ECDSA, RSA-FDH/PSS) to the use of more sophisticated privacy-enhancing schemes such as group signatures.

Figure 3: Abstract components of the FLEXAUTH tool.

Selective Disclosure Component. This component (Selective Disclosure) takes some signed/certified message and produces a selective disclosure token. Depending on the service, the selective disclosure token may have a different representation. For instance, in case of malleable signatures, given a policy selective disclosure triggers an *redaction or sanitization process* and the selective disclosure token represents an updated signature for the non-redacted and/or updated information from the original message, which can be presented to receivers. Here it is important to note that this component does not require any interaction with or knowledge of secrets of the Auth component (i.e., does not require the secret of the originator of the document, i.e., from the signer). Alternatively, the Selective Disclosure component may take an arbitrary fresh message together with some secret user information (corresponding to some well defined group of users) and the selective disclosure token produced by the component certifies anonymous membership in the respective group and at the same time certifies the accompanied message.

Verification Component. Finally, we have a verification component (Verify), which obtains a selective disclosure token produced by the Selective Disclosure component and verifies its authenticity. Thereby, this component may provide different types of veri-

fication procedures. When using malleable signatures that rely on conventional digital signatures or group signatures one obtains public verifiability (such that everyone can verify the authenticity). Moreover, when relying on specific schemes, there may also be other modes of verification such as for designated verifiers only (here only designated verifiers will be convinced of the authenticity of the received documents but can not convince any other party of their authenticity).

2.4 Software Implementation

The libraries representing the FLEXAUTH tool are implemented in Java and, from a design point of view, inspired by the Java Cryptography Architecture. While the design of the libraries is detailed later in this document, we refer the reader to Deliverable D.6.5 for further implementation details and examples on how to use the libraries.

2.5 Services Based on FLEXAUTH

Subsequently, we briefly describe the two services based on the FLEXAUTH tool that are implemented within PRISMACLOUD.

Selective Authentic Exchange Service. This PRISMACLOUD service enables users to move their authentic documents to a cloud service and then delegate the sharing of selective but authentic parts of these documents to other parties (who can verify their authenticity). Thereby, this services realizes end-to-end authenticity in a sense that verifiers can be ensured that the data comes from the original data source and the cloud has only performed allowed modifications. But still, non-revealed information stays concealed. Moreover, while the protection of the authenticity takes place in the beginning of the process by the originator, the storage, selection and verification no longer require interactions with (or secrets from) the originator. Thereby this service also tackles problems like vendor lock-in or the need of the originator to provide these services and empowers the user to freely choose where to store the authentic data without loosing the flexibility of selective authentic exchange.

Privacy Enhancing ID Management Service. This PRISMACLOUD service implements privacy enhancing identity-management. In particular, it allows users to obtain a signature from some authority and then selective disclosure tokens (signatures) for the respective user can be computed. Verifiers are then any party that needs to be assured of the authenticity of a user and will so by being presented a selective disclosure token. A prime functionality of this service is that the selective disclosure token does not reveal the user's identity, but only attests membership to some group.

2.6 Terms and Definitions

Subsequently, we define terms that are used within the description of the FLEXAUTH tool.

2.6.1 Signer

Party that specifies admissible changes for a given message and uses the private singing key to produce a redactable signature for the message supporting the given admissible changes.

2.6.2 Redactor/Sanitizer

Party that when given a signature, the corresponding message and redaction/sanitization information represented via modification instructions produces a redacted/sanitized signature for the redacted/sanitized message obtained by applying the modification instructions to the given message.

2.6.3 Issuer

Party that holds a issuing key and issues a membership certificate to a user which certifies the membership in a group.

2.6.4 Opener

Party that holds an opening key and when given a group signatures (a selective disclosure token) is outputs the identity of the signer (producer of the selective disclosure token) and potentially a proof certifying this fact.

2.6.5 Linker

Party that holds a linking key and when given two group signatures (selective disclosure tokens) this party outputs whether both have been produced by the same anonymous signer or not.

2.6.6 Verifier

Party that is given a (potentially redacted/sanitized within the scope of the admissible changes) signature, the corresponding message and the signers' public verification key and checks whether the given signature is a valid signature for the given message under the given verification key.

2.6.7 Message

A bit string representing the information to be signed, where the bit string may represent an encoding of an arbitrary document format.

2.6.8 Accumulator

A cryptographic accumulator allows to accumulate a finite set of values into a single succinct value called accumulator. For every accumulated value, one can efficiently compute a witness, which certifies its membership in the accumulator. However, it is computationally infeasible to find a witness for any non-accumulated value.

2.6.9 Group

A (dynamic) group of users. In our context this group is associated to a group verification key and every member of the group can anonymously issue group signatures (selective disclosure tokens) on behalf of the group.

2.6.10 Redaction

Process in which the redactor produces a redacted signature of a given signature-message pair according to given modification instructions. The input to the redaction process may be an original signature-message pair or a signature-message pair obtained via the (multiple) application of a redaction process.

2.6.11 Sanitization

Process in which the sanitizer with the help of the sanitization key produces a sanitized signature of a given signature-message pair according to a given modification instructions. The input to the sanitization process may be an original signature-message pair or a signature-message pair obtained via the (multiple) application of a sanitization process.

2.6.12 Signature Scheme

A scheme that allows to produce signatures for messages using a secret signing key and given a signature, a message and a public verification key to check its validity.

2.6.13 Signature

A piece of information produced using a signing key and attesting the authenticity of a message.

2.6.14 Group Signature

A piece of information produced using a signing key and attesting the authenticity of a message. The authenticity is with respect to a group of users instead of a single signer and actually does not disclose the actual signer.

2.6.15 Selective Disclosure Token

A synonym for a group signature.

2.6.16 Redacted Signature

A signature produced by a redactable signature scheme that has been processed by a redactor or several redactors (one after the other) running the redaction process with respect to some non-trivial modification instructions.

2.6.17 Sanitized Signature

A signature produced by a sanitizable signature scheme that has been processed by a sanitizer (via a sequence of sanitizations) running the sanitization process with respect to some non-trivial modification instructions and the sanitization key.

2.6.18 Group Signature Scheme

A signature schemes that allows users to anonymously sign messages on behalf of a group of users.

2.6.19 Redactable Signature Scheme

A signature scheme that supports redaction.

2.6.20 Sanitizable Signature Scheme

A signature scheme that supports sanitization.

2.6.21 Admissible Changes

Admissible changes describe all allowed modifications of a message to be signed with a redactable/sanitizable signature scheme that can be applied within the redaction/sanitization process without invalidating the redactable/sanitizable signature. Admissible

changes are called non-trivial, if the admissible changes allow at least one modification of the original message yielding a redacted/sanitized message being different from the original message.

2.6.22 Modification Instructions

Instructions that describe the message redaction/sanitization, i.e., how a message is going to be redacted/sanitized by the reactor/sanitizer within a redaction/sanitization process. Modification instructions are called non-trivial, if the given message and the redacted/sanitized message are not identical.

2.6.23 Redacted Message

A message that is the output of the redaction process. It represents a message that is derived from a given message using modification instructions (that are compatible with the admissible changes specified during the process of producing the redactable signature for the original message).

2.6.24 Sanitized Message

A message that is the output of the sanitization process. It represents a message that is derived from a given message using modification instructions (that are compatible with the admissible changes specified during the process of producing the sanitizable signature for the original message) when given a sanitization key.

2.6.25 Verification key

A public information that corresponds to a secret signing key and allows the verification of signatures.

2.6.26 Signing key

A secret information that is used to produce signatures for messages. This key may be of different types depending on the signature scheme. It can be a signing key for a conventional, a redactable, a sanitizable or a group signature scheme.

2.6.27 Sanitization key

A secret information used by the sanitizer to produce sanitized signatures within a sanitizable signature scheme.

2.6.28 Issuing Key

A secret information that is used to issue membership certificates for group membership of users.

2.6.29 Opening Key

A secret information that is used by the opener to identify the producer of a group signatures (selective disclosure token).

2.6.30 Linking Key

A secret information that is used by the linker to decide whether two group signatures (selective disclosure tokens) have been produced by the same anonymous user or not.

2.6.31 Controllable Linkability

A property of a group signature scheme that introduces an additional party (the linker) that when given a linking key can decide whether group signatures (selective disclosure tokens) have been produced by the same anonymous user.

2.7 Malleable Signatures Library

Now, we are going to present the malleable signatures library. Therefore, we start with an abstract description of the high level cryptographic building blocks and then proceed with describing the architecture of the library in detail.

2.7.1 Abstract Description

Redactable Signatures. A redactable signature scheme (RSS) allows any party to *remove* parts of a signed message M (according to some policy specified during signing) using a so called redaction algorithm. This redaction algorithm outputs an updated signature for the modified message which still verifies under the original signer's public key pk. The important point is that this can be done without the signers' secret key sk.

In the following, we present the abstract description of redactable signatures as given in [DPSS15].

Before presenting the algorithmic interface, we need to introduce some notation. We assume that a message M is some arbitrarily structured piece of data, ADM is an abstract data structure which describes the admissible redactions and may contain descriptions of dependencies, fixed elements or relations between elements. MOD is used to actually describe how a message M is redacted. Next, we define how ADM, MOD and the message

M are tangled, for which we introduce the following notation: $\mathsf{MOD} \overset{\mathsf{ADM}}{\preceq} M$ means that MOD is a valid redaction description with respect to ADM and M. $\mathsf{ADM} \preceq M$ denotes that ADM matches M, i.e., ADM is valid with respect to M. By $M' \xleftarrow{\mathsf{MOD}} M$, we denote the derivation of M' from M with respect to MOD. Clearly, how MOD, ADM, $\overset{\mathsf{ADM}}{\preceq}$, $\xleftarrow{\mathsf{MOD}}$ and $\preceq$ are implemented depends on the data structure in question and on the features of the concrete RSS. Let us give a simple example for sets without using dependencies or other advanced features: then, MOD and ADM, as well as M, are sets. A redaction $M' \xleftarrow{\mathsf{MOD}} M$ simply would be $M' \leftarrow M \setminus \mathsf{MOD}$. This further means that $\mathsf{MOD} \overset{\mathsf{ADM}}{\preceq} M$ holds if $\mathsf{MOD} \subseteq \mathsf{ADM} \subseteq M$, while $\mathsf{ADM} \preceq M$ holds if $\mathsf{ADM} \subseteq M$. We want to stress that the definitions of these operators also define how a redaction is actually performed, e.g., if a redacted block leaves a visible special symbol $\perp$ or not.

Now, we are ready to present an algorithmic description of RSS.

Definition 2.1. *An* RSS *is a tuple of four efficient algorithms* (KeyGen, Sign, Verify, Redact)*, which are defined as follows:*

$\mathsf{KeyGen}(1^\lambda)$: *On input of a security parameter λ, this probabilistic algorithm outputs a keypair* $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{Sign}(\mathsf{sk}, M, \mathsf{ADM})$: *On input of a secret key* sk*, a message M and* ADM*, this (probabilistic) algorithm outputs a message-signature pair (M, σ) together with some auxiliary redaction information* red.[1]

$\mathsf{Verify}(\mathsf{pk}, \sigma, M)$: *On input of a public key* pk*, a signature σ and a message M, this deterministic algorithm outputs a bit $b \in \{0, 1\}$.*

$\mathsf{Redact}(\mathsf{pk}, \sigma, M, \mathsf{MOD}, \mathsf{red})$: *This (probabilistic) algorithm takes a public key* pk*, a valid signature σ for a message M, modification instructions* MOD *and auxiliary redaction information* red *as input. It returns a redacted message-signature pair (M', σ') and an updated auxiliary redaction information* red′.[2]

We also require that Sign returns $\perp$, if $\mathsf{ADM} \not\preceq M$, while Redact also returns $\perp$, if $\mathsf{MOD} \overset{\mathsf{ADM}}{\not\preceq} M$. Note that red can also be $\emptyset$ if no auxiliary redaction information is required.

Moreover, other schemes may add additional algorithms. As an example take the updatable and mergeable redactable signature scheme from [PS14]. This scheme got implemented for PRISMACLOUD as an Java crypto provider to extend FLEXAUTH, see PRISMACLOUD Deliverable D6.5 for further details. Note, contrary to the above scheme it only protects sets not lists, i.e., M becomes a set S. Of course this means that the scheme can not protect the order of elements nor the cardinality of duplicates. Further note, the scheme tracks and identifies each individual signed message via a tag τ. For brevity we only give the scheme's additional algorithms and defer the interested reader to the original publication [PS14]. They are named Update and Merge and got defined as follows:

[1] We assume that ADM can always be correctly and unambiguously derived from any valid message-signature pair. Also note that ADM may change after a redaction.

[2] Note that this algorithm may either explicitly or implicitly alter ADM in an unambiguous way.

Definition 2.2. *The mergeable and updatable* UMRS *from [PS14] consists of six efficient algorithms. Let* UMRS := (*KeyGen*, *Sign*, *Verify*, *Redact*, *Update*, *Merge*). *In the following we only re-produce* Update *and* Merge *for brevity.*

Update : *The algorithm* Update *takes as input a verifying set/signature/tag tuple* $(\mathcal{S}, \sigma, \tau)$, *the secret key* sk *and a second set* $\mathcal{U}$. *It outputs* $(\mathcal{S}', \sigma', \tau)$, *where* $\mathcal{S}' = \mathcal{S} \cup \mathcal{U}$, *and* σ' *is a verifying signature on* $\mathcal{S}'$. *On error, the algorithm outputs* $\perp$

Merge : *The algorithm* Merge *takes as input the public key* pk *of the signer, two sets* $\mathcal{S}$ *and* $\mathcal{V}$, *a tag* τ, *and the corresponding signatures* $\sigma_{\mathcal{S}}$ *and* $\sigma_{\mathcal{V}}$. *We require that* $\sigma_{\mathcal{S}}$ *and* $\sigma_{\mathcal{V}}$ *are valid on* $\mathcal{S}$ *and* $\mathcal{V}$. *It outputs the merged set/signature/tag tuple* $(\mathcal{U}, \sigma_{\mathcal{U}}, \tau)$, *where* $\mathcal{U} = \mathcal{S} \cup \mathcal{V}$ *and* $\sigma_{\mathcal{U}}$ *is valid on* $\mathcal{U}$. *On error, the algorithm outputs* $\perp$

Sanitizable Signatures. Sanitizable signatures allow to issue a signature on a message where certain predefined message blocks may later be changed (sanitized) by some dedicated party (the sanitizer) who is identified by a public key. A signature updated by the sanitizer by means of the so called sanitizing algorithm (which requires the secret key of the sanitizer) does not invalidate the original signature of the signer as long as the modification are allowed (admissible). With sanitizable signatures, replacements for modifiable (admissible) message blocks can be chosen *arbitrarily* by the sanitizer. However, in various scenarios this makes sanitizers *too powerful*. To reduce the sanitizers power, one can rely on so called extended sanitizable signatures, where the most important extension (which we include subsequently) enables the signer to limit the allowed modifications per admissible block to a well defined set each.

In the following, we present an abstract description of extended sanitizable signature schemes ESSS, which cover sanitizable signatures as a special case, i.e., when omitting the extensions regarding `LimitSet` and ADM, it is equivalent to sanitizable signatures as presented in [BFF+], which is generally considered as the standard model for sanitizable signature schemes. Our description is as given in [DS15].

Definition 2.3 (Message)**.** *A message* $\mathsf{m} = (\mathsf{m}_i)_{i=1}^n$ *is a sequence of* n *bitstrings (message blocks).*

Henceforth, we use ℓ_i to refer to the (maximum) length of message block m_i and assume an encoding that allows to derive $(\ell_i)_{i=1}^n$ from m.

Definition 2.4 (Admissible Modifications)**.** *Admissible modifications* ADM *with respect to a message* $\mathsf{m} = (\mathsf{m}_i)_{i=1}^n$ *are represented as a sequence* $\mathsf{ADM} = (\mathsf{B}_i)_{i=1}^n$, *with* $\mathsf{B}_i \in \{\texttt{fix}, \texttt{var}, \texttt{lim}\}$.

Here $\mathsf{B}_i = \texttt{fix}$ indicates that no changes are allowed, $\mathsf{B}_i = \texttt{var}$ indicates that arbitrary replacements are allowed, and $\mathsf{B}_i = \texttt{lim}$ indicates that the replacements are limited to a predefined set (`LimitSet`).

Definition 2.5 (Set Limitations)**.** *Set limitations* V *with respect to a message* $\mathsf{m} = (\mathsf{m}_i)_{i=1}^n$ *and admissible modifications* $\mathsf{ADM} = (\mathsf{B}_i)_{i=1}^n$ *are represented by a set* $\mathsf{V} = \{(i, \mathsf{M}_i) : \mathsf{B}_i = \texttt{lim} \ \wedge \ \mathsf{M}_i \subset \bigcup_{j=0}^{\ell_i} \{0,1\}^j\}$.

We use $\mathsf{m}' \preceq^{\mathsf{ADM}} \mathsf{m}$ to denote that m' can be derived from m under ADM and V.

Definition 2.6 (Witnesses)**.** *Witnesses* $\mathcal{W} = \{(i, \mathcal{W}_i)\}_{i=1}^{t}$*, with* $\mathcal{W}_i = \{(m_{i_1}, \mathsf{wit}_{i_1}), \ldots, (m_{i_k}, \mathsf{wit}_{i_k})\}$*, are derived from set limitations* $\mathsf{V} = \{(i, \mathsf{M}_i)\}_{i=1}^{t}$*, with* $\mathsf{M}_i = \{m_{i_1}, \ldots, m_{i_k}\}$*. Thereby,* wit_{i_j} *attests that its corresponding message block* m_{i_j} *is contained in the set* M_i*.*

With $\mathcal{V} \xleftarrow{\mathsf{MOD}} \mathcal{W}$, we denote the extraction of the set of witnesses $\mathcal{V}$ corresponding to a message m from the set $\mathcal{W}$.

Definition 2.7 (Modification Instructions)**.** *Modification instructions* MOD*, with respect to a message* $\mathsf{m} = (\mathsf{m}_i)_{i=1}^{n}$*, admissible modifications* ADM *and set limitations* V *are represented by a set* $\mathsf{MOD} = \{(i, \mathsf{m}'_i)\}_{i=1}^{t}$ *with* $t \leq n$*, where* i *refers to the position of the message block in* m*, and* m'_i *is the new content for message block* m_i*.*

With $\mathsf{MOD} \preceq (\mathsf{ADM}, \mathsf{V})$, we denote that the modification instructions in MOD are compatible with ADM and V. Furthermore, with $(\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{ADM}, \mathsf{V}) \equiv (\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{ADM}, \mathsf{V})$, we denote that after applying the changes in MOD_0 and MOD_1 to m_0 and m_1 respectively, the resulting messages m'_0 and m'_1 are identical.

Definition 2.8. *An* ESSS *is a tuple of PPT algorithms* $(\mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *which are defined as follows:*

$\mathsf{KeyGen}_{\mathsf{sig}}(1^{\kappa})$**:** *This algorithm takes as input a security parameter* κ *and outputs a keypair* $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$ *for the signer.*

$\mathsf{KeyGen}_{\mathsf{san}}(1^{\kappa})$**:** *This algorithm takes as input a security parameter* κ *and outputs a keypair* $(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}})$ *for the sanitizer.*

$\mathsf{Sign}(\mathsf{m}, \mathsf{ADM}, \mathsf{V}, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}})$**:** *This algorithm takes as input a message* m*, corresponding admissible modifications* ADM *and set limitations* V*, as well as the keypair* $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$ *of the signer and the verification key* $\mathsf{pk}_{\mathsf{san}}$ *of the sanitizer. It computes the set* $\mathcal{W}$ *from* V*, obtains* $\mathcal{V} \xleftarrow{\mathsf{MOD}} \mathcal{W}$ *and outputs a signature* $\sigma = (\hat{\sigma}, \mathcal{V})$ *together with some auxiliary sanitization information* $\mathsf{san} = (\mathsf{aux}, \mathcal{W})$*.*[3] *In case of an error,* $\perp$ *is returned. As in [BFF+], we assume that* ADM *can be recovered from a signature* σ*.*

$\mathsf{Sanit}((\mathsf{m}, \sigma), \mathsf{MOD}, \mathsf{san}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$**:** *This algorithm takes as input a valid message-signature pair* (m, σ)*, modification instructions* MOD*, some auxiliary sanitization information* san *and the verification key* $\mathsf{pk}_{\mathsf{sig}}$ *of the signer and the signing key* $\mathsf{sk}_{\mathsf{san}}$ *of the sanitizer. It modifies* m *and* σ *according to* MOD *and outputs an updated message-signature pair* (m', σ') *and* $\perp$ *if* $\mathsf{m}' \not\preceq^{\mathsf{ADM}} \mathsf{m}$*. We assume that* V *can be reconstructed from* san*.*

$\mathsf{Verify}((\mathsf{m}, \sigma), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$**:** *This algorithm takes as input a message-signature pair* (m, σ) *and the public verification keys of the signer* $\mathsf{pk}_{\mathsf{sig}}$ *and the sanitizer* pksan*. It returns* `true` *if* σ *is a valid signature on* m *under* $\mathsf{pk}_{\mathsf{sig}}$ *and* $\mathsf{pk}_{\mathsf{san}}$*, and* `false` *otherwise.*

[3] While san is not required for plain sanitizable signature schemes, ESSS additionally return san to pass auxiliary information, which is only relevant for the sanitizer.

$\mathsf{Proof}((\mathsf{m},\sigma),\{(\mathsf{m}_j,\sigma_j)\}_{j=1}^{q},(\mathsf{sk},\mathsf{pk})\mathsf{sig},\mathsf{pksan})$**:** *This algorithm takes as input a message-signature pair* (m,σ)*, q message-signature pairs* $\{(\mathsf{m}_j,\sigma_j)\}_{j=1}^{q}$ *created by the signer, the keypair* $(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}})$ *of the signer and the public key* $\mathsf{pk}_{\mathsf{san}}$ *of the sanitizer and outputs a proof* π*.*

$\mathsf{Judge}((\mathsf{m},\sigma),\mathsf{pksig},\mathsf{pksan},\pi)$**:** *This algorithm takes as input 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 proof* π*. It outputs a decision* $d \in \{\mathtt{sig},\mathtt{san}\}$*, indicating whether the signature has been produced by the signer or the sanitizer.*

2.7.2 Architecture and Design of the Library

In this section, we map the abstract description given above to a concrete architecture in terms of components, and also provide more detailed insights in terms of class diagrams of the single components. Our library is split into four components which we will discuss subsequently.

MSS Provider. The MSS provider can be registered as a Java cryptographic provider. It uses the Java cryptography architecture (JCA) to provide an easy and interoperable way to generate certain parameters required by other components within our library and/or by external components. Currently, the MSS provider provides a `KeyPairGenerator` implementation to generate t-SDH instances as required by our accumulator implementation (see below). Due to the modularity which is inherited from the Java cryptography provider an easy extension is possible at a later point in time. Since this component only implements interfaces from the JCA, we omit a detailed description and present a design overview in Figure 4.

Figure 4: MSS Provider Class Diagram

Signature Engine. To abstract away the way how the malleable signature implementations generate the required standard digital signatures, we introduced another layer for the signature generation (cf. Figure 5). Essentially, the idea is to introduce a proxy `IDSSProxy` which provides means to obtain signing keypairs and to generate and verify signatures. Currently we provide means to generate signatures using the JCA. In particular, our implementations currently allow to create RSA and ECDSA signatures. At a later point in time, one can simply implement this interface to forward the calls to signature generation hardware as, e.g., developed within WP6.

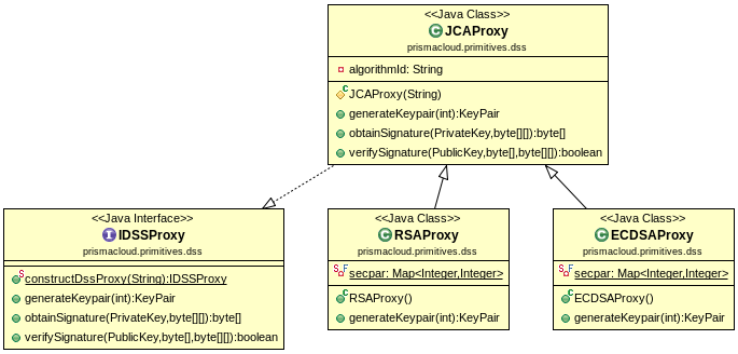

Figure 5: Signature Engine Class Diagram

Accumulator Framework. One essential building block of both redactable and extended sanitizable signatures are cryptographic accumulators. To this end, one component is dedicated to provide an easily extensible framework for accumulators. As a starting point for our implementation, we used the recent unification of cryptographic accumulators [DHS], which is also used as a building block in both the generic redactable signature framework [DPSS15] and the extended sanitizable signature framework [DS15] we are aiming to implement. Building upon this unified framework allows to easily exchange the concrete accumulator scheme used in implementations, i.e., one simply needs to implement the `IStatelessAccumulator` interface which resembles the abstract model from [DHS]. The factory method `IStatelessAccumulator.constructAccumulatorByClassName` is a generic factory method, which allows to obtain new instantiations of such implementations by simply providing the full-qualified class name.[4] The `IStatelessAccumulator` interface is wrapped by the class `IStatefulAccumulator` which—if required—takes care of maintaining a state. Currently we provide an implementation of the indistinguishable t-SDH accumulator from [ACN13, DHS].

Redactable Signature Framework. The redactable signature framework follows a similar design rationale as the accumulator framework and provides a `GenericRSS` class which closely follows the interface definition from [DPSS15] as presented above. The factory method `GenericRSS.constructRSS` allows to obtain an instantiation of a concrete `GenericRSS` implementation by its full-qualified class name, the full-qualified class

[4] Note that this design strategy resembles the design strategy of the JCA.

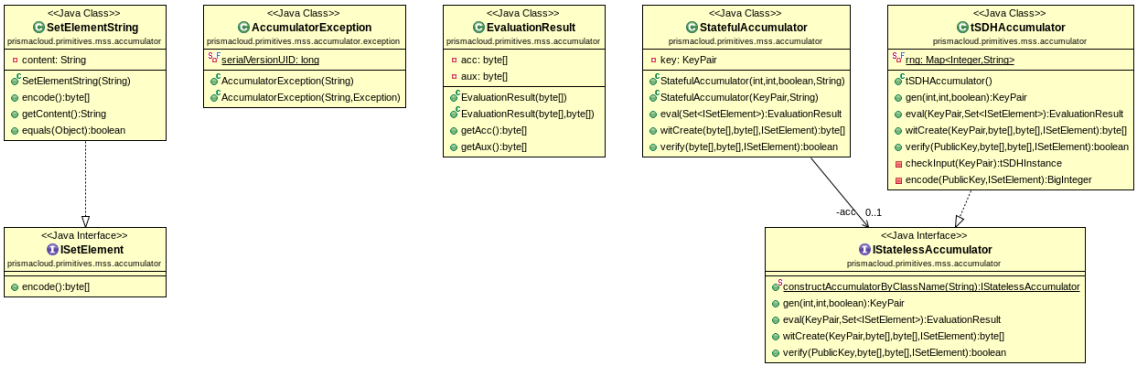

Figure 6: Accumulator Framework

name of the `IDSSProxy` implementation, as well as the full-qualified class name of the `IStatelessAccumulator` implementation. This design facilitates extreme flexibility with respect to the used underlying building blocks and signing mechanisms, i.e., one can simply exchange the whole accumulator or conventional signing procedure by simply modifying the class name passed to the factory method. Currently, we provide an implementation

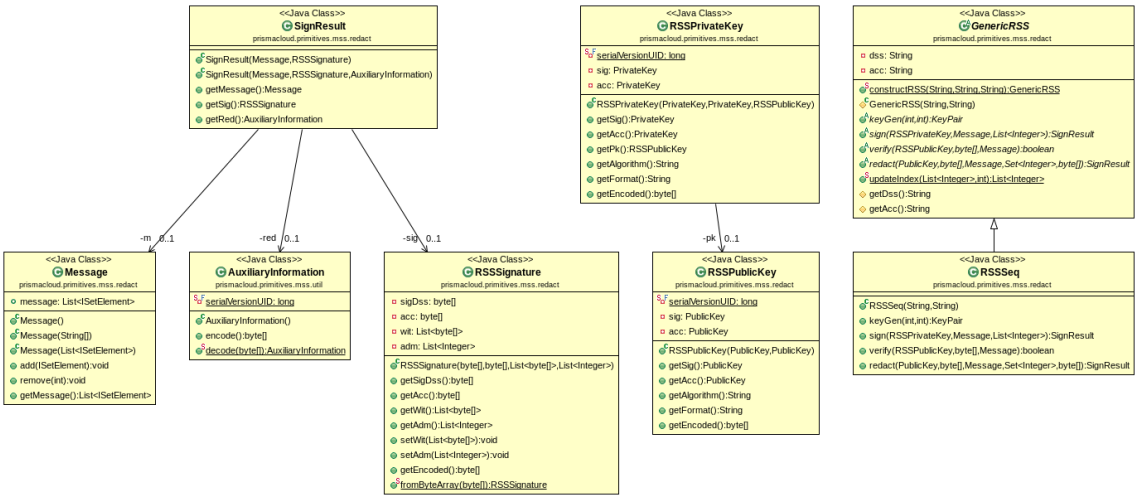

Figure 7: Redactable Signature Framework Class Diagram

of a generic redactable signature scheme for linear documents, i.e., documents which can be represented as a sequence of message blocks. While this is a format which will directly be suited for many applications including the use cases required within PRISMACLOUD, we stress that one can simply extend the `Message` class to implement the support for custom formats. Note that the scope of this document is the architecture of the FLEX-AUTH tool; further details on the concrete implementation were already provided in a preliminary version in Deliverable D6.5 and the final version will be provided in D6.6 in M30.

Extended Sanitizable Signature Framework. Currently, (extended) sanitizable signatures are not required within the scope of PRISMACLOUD. Yet, we believe that it is important to have a design which facilitates future developments and potential additional requirements which may arise over time. To this end we also evaluated the compatibility of our design with extended sanitizable signatures. Subsequently, we briefly sketch how extended sanitizable signatures would be integrated. Essentially we can use a design analogous to the redactable signature component, but tailored to the extended sanitizable signature interface [DS15]. In addition, sanitizable signatures [BFF+09] underlying the extension from [DS15] rely on chameleon hashes [BFF+09]. Thus we would require one additional component implementing chameleon hashes, where the design is in the same fashion as our accumulator component.

2.8 Group Signatures Library

Subsequently, we present the group signatures library. As with the former library, we start with an abstract description of the high level cryptographic building blocks and then proceed with describing the architecture of the library in detail.

2.8.1 Abstract Description

Group Signatures with Controllable Linkability. Group signatures allow users to join some group (by getting a signed membership certificate) and then to anonymously authenticate as a member of the group. Together with this authentication, a user can sign/authenticate any messages on behalf of a group (produce a so called group signature). In case of dispute, a so-called opening authority is able to reveal the identity of an anonymous user. Controllable linkability is a rather recent feature of group signatures. Here, a dedicated entity called linking authority (LA) can determine whether two given group signatures stem from the same user, but the LA is *not* able to identify the user(s).

Subsequently, we define the algorithms constituting a group signature scheme with controllable linkability (CL-GS), where we follow the presentation in [SSU14, BDSS16].

Definition 2.9. *A CL-GS is a tuple of efficient algorithms* $\mathcal{GS} = (\mathsf{GkGen}, \mathsf{UkGen}, \mathsf{Join}, \mathsf{Issue}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open}, \mathsf{Judge}, \mathsf{Link})$*, which are defined as follows.*

$\mathsf{GkGen}(1^\lambda)$: *On input a security parameter* λ*, this algorithm generates and outputs a tuple* $(\mathsf{gpk}, \mathsf{mok}, \mathsf{mik}, \mathsf{mlk})$*, representing the group public key, the master opening key, the master issuing key, and the master linking key.*

$\mathsf{UkGen}(1^\lambda)$: *On input a security parameter* λ*, this algorithm generates a user key pair* $(\mathsf{usk}_i, \mathsf{upk}_i)$*.*

$\mathsf{Join}(\mathsf{usk}_i, \mathsf{upk}_i)$: *On input the user's key pair* $(\mathsf{usk}_i, \mathsf{upk}_i)$*, this algorithm interacts with* Issue *and outputs the group signing key* gsk_i *of user* i*.*

$\mathsf{Issue}(\mathsf{gpk}, \mathsf{mik}, \mathbf{reg})$: *On input of the group public key* gpk*, and the master issuing key* mik *and the registration table* **reg***, this algorithm interacts with* Join *to add user* i *to the group.*

$\mathsf{GSig}(\mathsf{gpk}, M, \mathsf{gsk}_i)$: *On input of the group public key* gpk*, a message* M*, and a user's secret key* gsk_i*, this algorithm outputs a group signature* σ*.*

$\mathsf{GVf}(\mathsf{gpk}, M, \sigma)$: *On input of the group public key* gpk*, a message* M*, and a signature* σ*, this algorithm verifies whether* σ *is valid with respect to* M *and* gpk*. If so, it outputs* 1 *and* 0 *otherwise.*

$\mathsf{Open}(\mathsf{gpk}, \mathbf{reg}, M, \sigma, \mathsf{mok})$: *On input of the group public key* gpk*, the registration table* ***reg****, a message* M*, a valid signature* σ*, and the master opening key* mok*, this algorithm returns the signer* i *together with a publicly verifiable proof* τ *attesting the validity of the claim. If no group member produced* σ*,* $\perp$ *is returned.*

$\mathsf{Judge}(\mathsf{gpk}, M, \sigma, i, \mathsf{upk}_i, \tau)$: *On input of the group public key* gpk*, a message* M*, a valid signature* σ*, the claimed signer* i*, the public key* upk_i *as well as a proof* τ*, this algorithm returns* 1 *if* τ *is a valid proof that* i *produced* σ *and* 0 *otherwise.*

$\mathsf{Link}(\mathsf{gpk}, M, \sigma, M', \sigma', \mathsf{mlk})$: *On input of the group public key* gpk*, a message* M*, a corresponding valid signature* σ*, a message* M'*, a corresponding valid signature* σ' *and the master linking key* mlk*, this algorithm determines whether* σ *and* σ' *have been produced by the same or different signers and returns the linking decision* $b \in \{1, 0\}$*.*

2.8.2 Architecture and Design of the Library

Again, we deem it to be most reasonable to align our interfaces with existing models of group signatures from the literature which are broadly accepted and widely used. The model for controllably linkable group signatures given above, extends the most common model for dynamic group signatures from Bellare et al. [BSZ05], and, therefore, seems to be a good choice. As in the previous section, our `IGroupSignature` interface provides a factory method to obtain instance of `IGroupSignature` implementations via their full-qualified class name. The remaining methods basically resemble the interface methods as given in the abstract description above, or some helper methods which are required to construct instances of certain helper objects with respect to some parameters. Given the abstract interface description above, all methods are fairly self-explanatory and therefore not detailed. Besides, that group signatures also require the group manager to maintain a so-called registration table. We abstracted this table via a `IRegistrationTable` interface with associated factory method. This way our library is compatible with arbitrary registration table implementations, e.g., such an implementation could serve as a gateway to some registration table which is operated as a cloud service.

We currently provide implementations of two group signature schemes. First, we have implemented a group signature scheme which is due to Delerablée and Pointcheval [DP06]. Second, we have implemented the currently most efficient group signature scheme [DS16]. The latter scheme was developed within the context of PRISMACLOUD and is especially

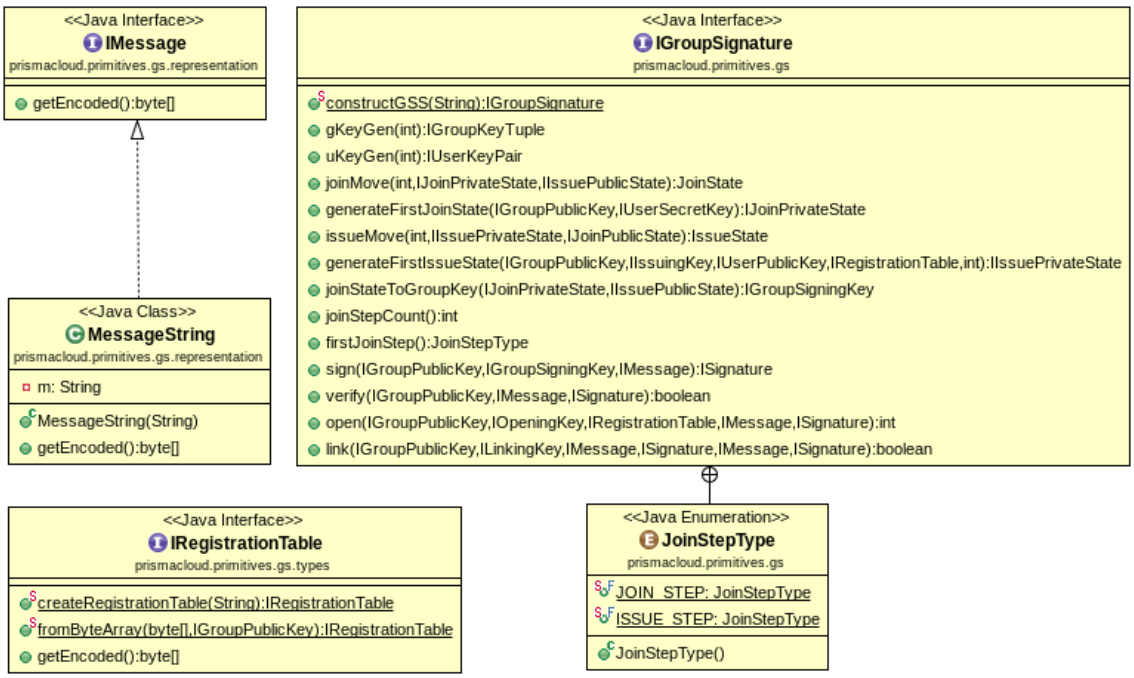

Figure 8: Group Signature Class Diagram

well-suited for the PRISMACLOUD use cases due to its efficiency. Note that the scope of this document is the architecture of the FLEXAUTH tool; further details on the implementation will be presented within D6.6 in M30.

Due to our modular architecture, one can use both signature schemes using the same code by simply exchanging the full qualified class name within the factory method. This makes it especially easy to switch between multiple group signature schemes, depending on the requirements of the specific use case where our tool is used.

2.9 Recommendations

Finally, we provide some recommendations on how to securely use the libraries. As seen above, both our libraries are constructed very generically and, apart from the concrete instantiations of primitives, can be easily extended by implementing additional primitives that fit into the frameworks. It is, however, important that the parameters are chosen in a way that the expected security from the primitives is guaranteed.

Choice of Key Sizes. For the ease of use our libraries are designed in a way that all cryptographic primitives can be instantiated by only providing a *security parameter*, which corresponds to the bit-strength of a symmetric cipher. This is a common way to compare the security of different primitives based on different cryptographic assumptions.

We, thereby, follow the NIST recommendations[5], which we present in Figure 9 for the convenience of the reader.

Date	Minimum of Strength	Symmetric Algorithms	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash (A)	Hash (B)
2010 (Legacy)	80	2TDEA*	1024	160	1024	160	SHA-1** SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
2011 - 2030	112	3TDEA	2048	224	2048	224	SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
> 2030	128	AES-128	3072	256	3072	256	SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
>> 2030	192	AES-192	7680	384	7680	384	SHA-384 SHA-512	SHA-224 SHA-256 SHA-384 SHA-512
>>> 2030	256	AES-256	15360	512	15360	512	SHA-512	SHA-256 SHA-384 SHA-512

Figure 9: NIST key length recommendations taken from https://www.keylength.com/en/4/.

For the parametrizations of the pairing-based primitives we have to consider suitable choices of pairing-friendly elliptic curve families for a deployment in practice. When choosing a suitable paring-friendly curve family, the recent advances for computing discrete logarithms in finite extension fields [KB16] applying to the target group $\mathbb{G}_T$ of state of the art pairing-friendly elliptic curve groups has to be taken into consideration. In this context, a recently released assessment of the impact of theses advances by Menezes et al. [MSS16] states 384 bit curves as a conservative estimate for BN curves [BN05]. In contrast, for BLS12 curves [BLS02] following Menezes et al. the advances have no impact at the current recommendations being 384 bit. However, recent work by Barbulescu et al. [BD17] raises the recommended bit lengths of the prime fields for the 128 bit security level to 461 bits for both BN and BLS12 curves.

[5] https://www.keylength.com/en/4/

3 The FLEXAUTH Tool in the Application Context

In this section, we first briefly set our architecture in the context of the pilot applications within PRISMACLOUD. Then we present our research on additional applications and also discuss how those applications would comply with the architecture of our framework.

3.1 The eHealth Pilot

Very roughly, in the eHealth use case, a hospital stores digital patient records within a cloud platform so that users can further distribute selected parts of their documents to further entities, e.g., to produce sick notes for employers or for further treatment by the family doctor. We aim to use the FLEXAUTH tool within this use case to allow doctors in the hospital to sign the patient records using an RSS. This way, one can remove certain document parts while still maintaining end-to-end authenticity and source authentication. Practically speaking, in our example from before the employer or the family doctor can then still be sure that they are indeed presented an authentic portion of the health record which was actually signed by a doctor in the hospital.

While further details on the implementation are provided within D7.8, we stress one important point in the design of our library which facilitates an easy integration within the hospital's existing infrastructure. That is, we provide an interface which abstracts the signature generation as well as the provisioning of the key material. As detailed in D7.8 this allows us an easy integration within the identity provisioning and signing framework provided by the partner XiTrust.

3.2 The Smart City Pilot

The Smart City pilot builds upon the results of the SIMON project[6], and, in particular, aims to address privacy and security issues identified within the SIMON project and in particular the disabled parking use-case. The functionality of the FLEXAUTH tool is thereby used to increase users' privacy. From a very high-level, users obtain group signing keys upon registration which enables them to issue signatures for a certain period (say a month). To park at a certain parking spot, they deposit a group signature on some message, so that the ticket inspector can verify whether they are actually from the group of authorized entities.[7] To detect "double-spending", i.e., find entities who use their signing key to park on two distinct parking spots, we use the linking feature of group signatures.

Using the defined interface, the integration of the FLEXAUTH tool within the use case is almost straight-forward. Also note that our flexible design allows to easily switch between different concrete implementations of group signature schemes.

[6] http://simon-project.eu

[7] Note that we employ some additional measures to prevent replay attacks which are omitted here for brevity. Refer to D7.8 for the details.

3.3 Research on Additional Applications

3.3.1 Cryptographically Enforced Four-Eyes Principle

In [BHPS16], we have shown that sanitizable signatures can be used to implement a well-known access control and authorisation concept: The 4-eyes principle (4EP). It is used in many workflows to minimise the likelihood of corruption. It states that at least two separate entities must approve a message before it is considered authentic. Hence, an adversarial party aiming to forge bogus content is forced to convince other parties to collude in the attack. In the aforementioned paper, we present a formal framework along with a suitable security model. Namely, a party sets a policy for a given message which involves multiple additional approvers in order to authenticate the message. Based on this, we show how such a signature can be black-box realised by secure sanitizable signature schemes and thus make the FLEXAUTH tool usable in such application scenarios.

For convenience of the reader, we have included a copy of [BHPS16] in the Appendix.

From an implementation point of view, we want to remark that the architecture of the FLEXAUTH tool is designed with extensibility in mind. In particular, as sketched in Section 2.7.2 it is easily possible to add support for (extended) sanitizable signature schemes.

3.3.2 Accountable and Privacy Preserving Workflows

In [DHPS15], we discuss applications of malleable signatures in the context of accountable and privacy preserving documentation of outsourced workflows. In particular, envision a business process where certain sub-tasks are outsourced to some tenant in the cloud. To efficiently manage those processes, to immediately detect deviations from the intended workflows and to hold tenants (such as the cloud provider) accountable in such (decentralised) 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 organisational measures. A first idea to hold the tenant accountable for its actions, would be to require the tenant to digitally sign a report documenting the performed operations. We however, observe that using variants of malleable signatures has benefits in these applications as it allows to increase the expressiveness of such reports in that it allows to predefine the structure of the reports and potentially even certain restrictions on the actions the tenant is allowed to perform. We demonstrate the usefulness of certain *variants* of malleable signature schemes, as well as *proxy (functional)* signature schemes, i.e., signature schemes which allow the *delegation of signing rights* to other parties, in this context.

For convenience of the reader, we have included a copy of [DHPS15] in the Appendix.

As above, we observe that the architecture of the FLEXAUTH tool is compatible with all those variants of signature schemes.

3.4 Extending Redactable Signatures with Additional Privacy Features

In [DKS16], we further extend redactable signatures with additional privacy features. In particular, we observe that there are scenarios where the identity of the signer already leaks privacy sensitive information. In addition, we observe that anyone who is in possession of a valid signature on a (redacted) document can hand it on to other parties to convince them of the authenticity of the data. However, it would be more privacy friendly to have (redactable) signatures which only convince a designated verifier of the authenticity of the associated document (as it is known from universal designated verifier signatures [SBWP03]). We extend the security model of redactable signatures in these directions and also present two provably secure constructions in the extended model.

For convenience of the reader, we have included a copy of [DKS16] in the Appendix.

From the practical viewpoint, our FLEXAUTH architecture would allow for an easy (and even generic) integration of the group signing feature by using the abstraction layer we introduced for the generation of digital signatures in our generic RSS interface. For the designated verifier feature, one would additionally require the implementation of the required zero-knowledge proofs of knowledge, which however is rather straight-forward.

4 Conclusion

In this deliverable we have presented our final design for the FLEXAUTH tool. We have established a flexible design with an easy to use API which is compatible with all the requirements imposed by the PRISMACLOUD pilots. While the design itself represents a final iteration of the design already presented in D5.6, we have presented additional research on applications going beyond the use cases within the PRISMACLOUD pilots, and assessed the possibilities of an integration of the primitives required by those extended application scenarios.

List of Acronyms

t-SDH	t Strong Diffie Hellman
4EP	Four-Eyes Principle
API	Application Programming Interface
BFT	Byzantine Fault Tolerance
CCTV	Closed-Circuit Television
CL-GS	Group Signature Scheme with Controllable Linkability
CryptSDLC	Cryptographic Service Development LiveCycle
CSS	Computational Secret Sharing
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
EUF-CMA	Existential Unforgeability under adaptively Chosen Message Attacks
FPE	Format Preserving Encryption
GSS	Group Signature Scheme
GS	Group Signature
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extensions
JCPF	Java Crypto Provider Framework
JSON	JavaScript Object Notation
LA	Linking Authority
MSS	Malleable Signature Scheme
OPE	Order Preserving Encryption
PSS	Proactive Secret Sharing
RSA-FDH	RSA Full Domain Hash
RSA-PSS	RSA Probabilistic Signature Scheme
RSS	Redactable Signature Scheme
VDP	Verifiable Data Processing
XML	eXtensible Markup Language

List of Figures

1 PRISMACLOUD architecture. 7
2 Library components of the FLEXAUTH tool. 10
3 Abstract components of the FLEXAUTH tool. 10
4 MSS Provider Class Diagram 20
5 Signature Engine Class Diagram 21
6 Accumulator Framework 22
7 Redactable Signature Framework Class Diagram 22
8 Group Signature Class Diagram 25

9 NIST key length recommendations taken from https://www.keylength.com/en/4/. 26

References

[ACN13] Tolga Acar, Sherman S. M. Chow, and Lan Nguyen. Accumulators and U-Prove Revocation. In *Financial Cryptography*, LNCS. 2013.

[BD17] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *IACR Cryptology ePrint Archive*, 2017:334, 2017.

[BDSS16] Olivier Blazy, David Derler, Daniel Slamanig, and Raphael Spreitzer. Non-Interactive Plaintext (In-)Equality Proofs and Group Signatures with Verifiable Controllable Linkability. In *Topics in Cryptology – CT-RSA 2016*, pages 127–143, 2016.

[BFF+] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *PKC 2009*, volume 5443 of *LNCS*.

[BFF+09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *PKC*, 2009.

[BHPS16] Arne Bilzhause, Manuel Huber, Henrich C. Pöhls, and Kai Samelin. Cryptographically enforced four-eyes principle. In *11th International Conference on Availability, Reliability and Security, ARES 2016, Salzburg, Austria, August 31 - September 2, 2016*, pages 760–767, 2016.

[BLS02] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, pages 257–267, 2002.

[BN05] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, pages 319–331, 2005.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of Group Signatures: The Case of Dynamic Groups. In *CT-RSA*, volume 3376 of *LNCS*, pages 136–153. Springer, 2005.

[DHPS15] David Derler, Christian Hanser, Henrich C. Pöhls, and Daniel Slamanig. Towards authenticity and privacy preserving accountable workflows. In *Privacy and Identity Management. Time for a Revolution? - 10th IFIP WG 9.2, 9.5, 9.6/11.7, 11.4, 11.6/SIG 9.2.2 International Summer School, Edinburgh, UK, August 16-21, 2015, Revised Selected Papers*, pages 170–186, 2015.

[DHS] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA 2015*, volume 9048 of *LNCS*.

[DKS16] David Derler, Stephan Krenn, and Daniel Slamanig. Signer-anonymous designated-verifier redactable signatures for cloud-based data sharing. In *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, pages 211–227, 2016.

[DP06] Cécile Delerablée and David Pointcheval. Dynamic Fully Anonymous Short Group Signatures. In *Progress in Cryptology – VIETCRYPT 2006*, pages 193–210, 2006.

[DPSS15] D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig. A general framework for redactable signatures and new constructions. In *ICISC*, pages 3–19, 2015.

[DS15] D. Derler and D. Slamanig. Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In *ProvSec*, pages 455–474, 2015.

[DS16] David Derler and Daniel Slamanig. Fully-anonymous short dynamic group signatures without encryption, 2016.

[KB16] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 543–571, 2016.

[MSS16] Alfred Menezes, Palash Sarkar, and Shashank Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. *IACR Cryptology ePrint Archive*, 2016:1102, 2016.

[PS14] H. C. Pöhls and K. Samelin. On updatable redactable signatures. In *ACNS*, volume 8479 of *LNCS*, pages 457–475. Springer, 2014.

[SBWP03] Ron Steinfeld, Laurence Bull, Huaxiong Wang, and Josef Pieprzyk. Universal designated-verifier signatures. In *ASIACRYPT*, 2003.

[SSU14] Daniel Slamanig, Raphael Spreitzer, and Thomas Unterluggauer. Adding Controllable Linkability to Pairing-Based Group Signatures for Free. In *ISC*, volume 8783 of *LNCS*, pages 388–400. Springer, 2014.

A Appendix

In the following we have attached the publications that relate to Section 3 of this deliverable.

Cryptographically Enforced Four-Eyes Principle

Arne Bilzhause
Chair of IT-Security & Institute of IT-Security and Security Law (ISL),
University of Passau, Passau, Germany
ab@sec.uni-passau.de

Manuel Huber
Fraunhofer Research Institute AISEC, Munich, Germany
manuel.huber@aisec.fraunhofer.de

Henrich C. Pöhls
Chair of IT-Security & Institute of IT-Security and Security Law (ISL),
University of Passau, Passau, Germany
hp@sec.uni-passau.de

Kai Samelin
IBM Research – Zurich,Rüschlikon, Switzerland &
TU Darmstadt, Darmstadt, Germany
ksa@zurich.ibm.com

Abstract

The 4-eyes principle (4EP) is a well-known access control and authorization principle, and used in many scenarios to minimize the likelihood of corruption. It states that at least two separate entities must approve a message before it is considered authentic. Hence, an adversarial party aiming to forge bogus content is forced to convince other parties to collude in the attack. We present a formal framework along with a suitable security model. Namely, a party sets a policy for a given message which involves multiple additional approvers in order to authenticate the message. Finally, we show how these signatures are black-box realized by secure sanitizable signature schemes.

1 Introduction

Involving more than one party in important decisions or transactions is one way of fighting corruption or just making sure that accidental errors do not get overlooked. Using cryptography this can be expressed by requiring more than one valid digital signature on a message m, such that a verifier is assured that several distinct entities agreed on the same message. Only if all signatures verify at the same time, the message is considered valid. Let us

make an example; the signed message m is a PhD diploma issued by the faculty, and a valid diploma requires two professors as approvers. Hence, the diploma is only considered genuine, if two professors appointed by the dean of the faculty also approve it, i.e., have also signed it. Only if all signatures verify under the given (trusted) public keys, the diploma is considered genuine, meaning that each of the two professors agreed to graduate the student, and the faculty authorized them as approvers. Thus, neither a single professor, nor the dean of the faculty, has sufficient permissions to graduate the student in question on its own, which would require to corrupt the dean of the faculty and the two professors. Generally speaking, this is widely known as the four-eyes principle (4EP). We focus on the case where the policy requires two *additional* approvers. We choose this definition of the 4EP, as the messages to be approved may come from any source. In the given example, this could be a university server compiling the data of the PhD candidate in an automated fashion from several databases located in a partially untrusted cloud. In this scenario, the 4EP requires two *additional* entities to be meaningful. Clearly, this extends to other protocols where data is processed automatically, but still needs to be approved. Another example clarifying this statement is the external production of a payroll, where an external server located in the cloud prepares the checks, while two employees from human resources need to approve the calculations locally before they become valid and the bank would allow them to be cashed-in.

An alteration of our construction, and framework, to the "standard" 4EP with one approver is straightforward. Extensions to more than two approvers follow a simple pattern.

We show how sanitizable signature schemes (SSS) [3] can be used in this scenario. In a nutshell, SSSs allow to alter all signer-chosen admissible blocks $m[i]$ of a given message $m = (m[1], m[2], \ldots, m[i], \ldots, m[\ell])$ to different bitstrings $m[i]' \in \{0,1\}^*$, where $i \in \{1, 2, \ldots, \ell\}$, by a semi-trusted party named the sanitizer. This party holds its own private and public key. Thus, sanitization of a message m results in an altered message $m' = (m[1]', m[2]', \ldots, m[i]', \ldots, m[\ell]')$, where $m[i] = m[i]'$ for every non-admissible block, and also a signature σ', which verifies under the given public keys. We use this primitive to cryptographically enforce the 4EP. Our construction paradigm has the benefit that there is no need to agree on a set of participating parties beforehand, i.e., the entities do not need to know each other a-priori. Thus, all entities can generate their key pair in advance, without requiring to know which entity the other approver, or the entity generating the message, is. Moreover, our construction is completely non-interactive, meaning that neither for signing, nor for approving nor for verifying any interaction between parties is necessary. We also do not require that the message m is "tainted" with meta-information, i.e., the signature σ itself carries enough information to derive which parties are required to approve the message m in question, and also which party chose

the policy, and m. This is in particular useful for entities only allowed to approve messages, but not to generate messages to be approved, and vice versa. This principle separates concerns and means that professors are only allowed to approve a diploma, while only the dean of the faculty is allowed to generate diplomas, in our example.

1.1 Our Contribution

We introduce a formal framework that enforces the 4EP. This framework is accompanied by cryptographic security definitions, which capture the main idea of the 4EP. We then show how one can use the well-studied primitive of SSSs in this context. Namely, SSSs are enough to realize "4EP-signatures". We identify the necessary properties of SSSs, and present a provably secure construction meeting our requirements, black-box realized by any secure SSS. The reductions are tight, i.e., we only have a constant reduction loss, regardless of the security parameter length. Moreover, our construction paradigm has the advantage that the parties are not required to encode the approvers into the message m, which therefore also helps to separate concerns. In addition, we show how to extend this paradigm to more than two approving parties, and threshold schemes. Thus, we open new directions where SSSs perfectly fit in.

Note that our goal is not to exchange signatures between parties [2, 4], but to enforce entities to agree upon the same message m. In other words, not all entities are required to receive the approved signature σ, but only the final one. This allows to use less complex schemes.

1.2 State-of-the-Art

On the one hand, there exists work which can be used in our scenario as well. This includes multi-signatures [5, 6], aggregate signatures [7], threshold signatures [31], and proxy signatures [27]. However, all these primitives are either very complex compared to SSSs (aggregate signatures, and multi signatures), require some sort of interaction (proxy signatures), or the entities need to know each other a-priori (threshold signatures), or a trusted third party is involved. In our construction, the entity generating the message can decide ad-hoc (on a per message basis) which entities need to approve a given message. It also cannot forge signatures, i.e., if the appointed entities do not approve the message m, a verifier does not consider the signature σ valid.

On the other hand, SSSs have originally been introduced by *Ateniese* et al. [3]. *Brzuska* et al. formalized most of the current security properties in [8]. These have been later extended for unlinkability [10, 12], and non-interactive public accountability [11, 12]. Some properties have then been refined by *Gong* et al. [24]. Namely, they also consider the admissible blocks

in the security games. Recently, *Krenn* et al. further refined the security properties to also account for the signatures, not only the message [26]. Several extensions such as limiting the sanitizer to signer-chosen values [13, 21, 25, 30], trapdoor SSSs (which allow to add new sanitizers after signature generation by the signer) [15, 32], multi-sanitizer and -signer environments for SSSs [9, 12, 14], and sanitization of signed and encrypted data [22] have been considered. SSSs have also been used as a tool to make other related primitives accountable [29], and to build other primitives, such as redactable signatures schemes or credentials [16, 20, 18]. Also, SSSs and data-structures more complex than lists have been considered [30]. Several implementations of SSS presented in the literature prove that SSSs are sufficiently efficient for use in practices [11, 12, 17, 28, 30]. Refer to references [1, 19, 23] for a comprehensive overview of malleable signatures.

We stress that the SSSs we require can be built using standard unforgeable digital signatures. For example, the construction given by *Brzuska* et al. [11] is suitable for our needs. Thus, standard signatures are sufficient via a trivial construction. Namely, one requires three signatures on the message, one for the message generator, and one for each approver. Using SSSs, however, has the benefit that the primitive itself already offers the required interfaces, especially if existing implementations are re-used. Moreover, the roles of the entities are clearly separated, while our construction also allows for more efficient schemes, as two signatures are sufficient.

2 Preliminaries and Building Blocks

2.1 Notation

$\lambda \in \mathbb{N}$ denotes our security parameter. All algorithms implicitly take 1^λ as an additional input. We write $a \leftarrow A(x)$ if a is assigned the output of algorithm A with input x. For a message $m = (m[1], m[2], \ldots, m[\ell])$, where $m[i] \in \{0,1\}^*$, we call $m[i]$ a block, while $\ell \in \mathbb{N}$ denotes the number of blocks in a message m. An algorithm is efficient if it runs in probabilistic polynomial time (ppt) in the length of its input. The algorithms may return a special error symbol $\perp \notin \{0,1\}^*$, denoting an exception. For the remainder of this paper, all algorithms are ppt if not explicitly mentioned otherwise. If we have a list, we require that we have an injective encoding mapping the list to $\{0,1\}^*$. A message space $\mathcal{M}$, and the randomness space $\mathcal{R}$, may implicitly depend on the corresponding public key(s). If not otherwise stated, we assume that $\mathcal{M} = \{0,1\}^* \cup \perp$ to reduce unhelpful boilerplate notation, while $\mathcal{R}$ is implicit. A function $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible*, if it vanishes faster than every inverse polynomial, i.e., for every $k \in \mathbb{N}$ there exists an $n_0 \in \mathbb{N}$ such that $\nu(n) \leq n^{-k}$ for all $n > n_0$.

2.2 Sanitizable Signatures

The definitions are based on [8, 11, 12, 24, 26].

Definition 1 (Sanitizable Signature Schemes) *A sanitizable signature scheme* SSS *consists of seven ppt algorithms* $(\mathsf{KGen}_{\mathsf{sig}}, \mathsf{KGen}_{\mathsf{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *such that:*

1. *Signer Key Generation: The signer key pair generation creates a key pair for the signer; a private key and the corresponding public key, based on* λ*:* $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(1^{\lambda})$.

2. *Sanitizer Key Generation: The sanitizer key pair generation also returns a private key and the corresponding public key, based on* λ*, but for the sanitizer:* $(\mathsf{pk}_{\mathsf{san}}, \mathsf{sk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(1^{\lambda})$.

3. *Signing: The* Sign *algorithm takes as input a message* m*,* $\mathsf{sk}_{\mathsf{sig}}$*,* $\mathsf{pk}_{\mathsf{san}}$*, as well as a description* ADM *of the admissible blocks.* ADM *contains the set of indices of the modifiable blocks, as well as the number* ℓ *of blocks in* m*. We write* $\mathrm{ADM}(m) = \texttt{true}$*, if* ADM *is valid w.r.t.* m*, i.e.,* ADM *contains the correct* ℓ *and all indices are in* m*. If* $\mathrm{ADM}(m) = \texttt{false}$*, this algorithm returns* $\perp$*. For example, let* $\mathrm{ADM} = (\{1, 2, 4\}, 4)$*. Then,* m *must contain four blocks, while all but the third will be admissible. If we write* $m_i \in \mathrm{ADM}$*, we mean that* m_i *is admissible. It outputs a signature* $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathrm{ADM})$.

4. *Sanitizing: Algorithm* Sanit *takes a message* m*, modification instruction* MOD*, signature* σ*,* $\mathsf{pk}_{\mathsf{sig}}$*, and* $\mathsf{sk}_{\mathsf{san}}$*. It modifies the message* m *according to the modification instruction* MOD*, which is a set containing pairs* $(i, m[i]')$ *for those blocks that shall be modified, meaning that* $m[i]$ *is replaced with* $m[i]'$*.* Sanit *calculates a new signature* σ' *for the modified message* $m' \leftarrow \mathrm{MOD}(m)$*. It outputs* m' *together with* σ'*:* $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$*. We require that every party can always correctly derive which parts of the message* m *are admissible from any valid signature* σ*. This is in accordance with [8, 24].*

5. *Verification: The* Verify *algorithm outputs a decision* $d \in \{\texttt{true}, \texttt{false}\}$*, verifying the signature* σ *for a message* m *w.r.t. the public keys* $\mathsf{pk}_{\mathsf{sig}}$ *and* $\mathsf{pk}_{\mathsf{san}}$*:* $d \leftarrow \mathsf{Verify}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$.

6. *Proof: The* Proof *algorithm takes as input* $\mathsf{sk}_{\mathsf{sig}}$*, a message* m*, a signature* σ*, and a set of polynomially many additional message/signature pairs* $\{(m_i, \sigma_i)\}$ *and* $\mathsf{pk}_{\mathsf{san}}$*. It outputs a string* $\pi \in \{0,1\}^*$ *which can be used by the* Judge *to decide which party is accountable given a message/signature pair* (m, σ)*:* $\pi \leftarrow \mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk}_{\mathsf{san}})$.

7. *Judge: Algorithm* Judge *takes as input a message m, a signature σ, $\mathsf{pk}_{\mathsf{sig}}$, $\mathsf{pk}_{\mathsf{san}}$, as well as a proof π. Note, this means that once a proof π is generated, the accountable party can be derived by anyone for that message/signature pair (m, σ). It outputs a decision $d \in \{\mathsf{Sig}, \mathsf{San}\}$, indicating whether the message/signature pair has been created by the signer, or the sanitizer: $d \leftarrow \mathsf{Judge}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \pi)$.*

2.3 Correctness of Sanitizable Signature Schemes

We require the usual correctness requirements to hold. In a nutshell, every honestly signed, or sanitized, message/signature pair must verify, while an honestly generated proof on an honestly generated message/signature pair must point to the correct accountable party. Refer to [8] for a formal definition.

2.4 Security of Sanitizable Signature Schemes

Next, we introduce our security model. We only require a subset of the state-of-the-art properties [8, 12, 24]. Namely, we require immutability, and non-interactive public accountability. Our proofs of the construction can directly be reduced to these properties. Thus, we do not require unlinkability, privacy, or transparency. However, non-interactive public accountability implies signer-accountability, sanitizer-accountability, and unforgeability [11]. Moreover, we do not require the strong definitions given by *Krenn* et al. [26]. These definitions take also the signature σ itself into account, which is not necessary in our case.

2.5 Immutability

Clearly, a sanitizer must only be able to sanitize the admissible blocks defined by ADM. This also prohibits deleting, or appending blocks from a given message m. Moreover, the adversary is given full oracle access, while it is also allowed to generate the sanitizer key pair.

Definition 2 (Immutability) *An* SSS *is immutable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Immutability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda) = 1] \leq \nu(\lambda)$ *, where the corresponding experiment is defined in Fig. 1.*

2.6 Non-Interactive Public Accountability

Non-interactive public accountability allows everyone to decide whether a sanitizer was involved. This is modeled by requiring that Judge works with an empty proof, i.e., $\pi = \perp$. Hence, no secret keys are required to find the accountable party, and Proof can be defined as $\perp$.

Experiment $\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(1^\lambda)$
$(m^*, \sigma^*, \mathsf{pk}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot),\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{sig}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \mathsf{pk}_{\mathsf{san},i}, \mathrm{ADM}_i)$ index the queries to Sign
return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*) = \mathtt{true} \wedge$
$(\forall i \in \{1, 2, \ldots, q\} : \mathsf{pk}^* \neq \mathsf{pk}_{\mathsf{san},i} \vee$
$m^* \notin \{\mathrm{MOD}(m_i) \mid \mathrm{MOD} \text{ with } \mathrm{ADM}_i(\mathrm{MOD}) = 1\})$
return 0

Figure 1: Immutability

Experiment $\mathsf{Pubaccountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{san}}, \mathsf{sk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(1^\lambda)$
$(\mathsf{pk}^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}})}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \mathrm{ADM}_i, \mathsf{pk}_{\mathsf{san},i})$,
and σ_i index the queries/answers to/from Sign
for $j = 1, 2, \ldots, q'$ let $(m_j, \mathrm{MOD}_j, \sigma_j, \mathsf{pk}_{\mathsf{sig},j})$,
and (m'_j, σ'_j) index the queries/answers to/from Sanit
return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*) = \mathtt{true} \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*) \neq (\mathsf{pk}_{\mathsf{san},i}, m_i) \wedge$
$\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*, \perp) = \mathsf{Sig}$
return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{san}}) = \mathtt{true} \wedge$
$\forall j \in \{1, 2, \ldots, q'\} : (\mathsf{pk}^*, m^*) \neq (\mathsf{pk}_{\mathsf{sig},j}, m'_j) \wedge$
$\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{san}}, \perp) = \mathsf{San}$
return 0

Figure 2: Non-Interactive Public Accountability

Definition 3 (Non-Interactive Public Accountability) *An* SSS *is non-interactive publicly accountable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:* $\Pr[\mathsf{Pubaccountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 2.*

Definition 4 (Secure SSS) *We call an* SSS *secure, if it is correct, immutable, and non-interactive publicly accountable.*

We stress again, that we neither require unlinkability, transparency, nor privacy in our case, which may allow for more efficient realizations.

3 Cryptographically Enforcing the Four-Eyes Principle

In this section, we introduce the framework for signatures enforcing the 4EP. This includes suitable security definitions, which capture the main idea of the 4EP.

3.1 Our Framework

Our main idea is that a single party generates a message m, signs it, and asks for approval. Thus, after signature generation, two additional entities have to approve the message before it is considered valid by third parties, i.e., by verifiers. In particular, as the name already suggests, the approvers must only be able to approve a message m. Hence, an approver must not be able to generate or change the message without invalidating the signature.

Definition 5 (4EP-Signatures) *A signature scheme* 4EPSIG *enforcing the 4EP consists of five ppt algorithms, i.e.,* $(\mathsf{KGen}_{\mathsf{sig}}, \mathsf{KGen}_{\mathsf{App}}, \mathsf{Sign}, \mathsf{App}, \mathsf{Verify})$ *such that:*

1. *Signer Key Generation: The signer key pair generation creates a key pair for the signer; a private key and the corresponding public key, based on* λ*:* $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(1^{\lambda})$.
2. *Approver Key Generation: The approver key pair generation also returns a private key and the corresponding public key, based on* λ*, but for the approver(s):* $(\mathsf{pk}_{\mathsf{App}}, \mathsf{sk}_{\mathsf{App}}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^{\lambda})$*. If we have more than one key, we address them with a subscript.*
3. *Signing: The* Sign *algorithm takes as input a message* m*,* $\mathsf{sk}_{\mathsf{sig}}$*, and two approver public keys* $\mathsf{pk}_{\mathsf{App},1}$*, and* $\mathsf{pk}_{\mathsf{App},2}$*. It outputs a signature* $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\})$*. For easier analysis, we require that this algorithm returns* $\perp$*, if* $\mathsf{pk}_{\mathsf{App},1} = \mathsf{pk}_{\mathsf{App},2}$*. We also assume a canonical ordering of the set* $\{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}$*. We assume that* $\mathsf{pk}_{\mathsf{App},1}$ *denotes the "smallest" element in* $\{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}$*, denoted as* $\mathsf{pk}_{\mathsf{App},1} \prec \mathsf{pk}_{\mathsf{App},2}$.
4. *Approving: Algorithm* Approve *takes a message* m *to approve, a signature* σ*,* $\mathsf{pk}_{\mathsf{Sign}}$*, an approver public key* $\mathsf{pk}_{\mathsf{App}}$*, and an approver secret key* $\mathsf{sk}_{\mathsf{App}}$*. It approves the message* m *for the given parameters. Thus,* App *outputs a new, (potentially only partially) approved signature* $\sigma' \leftarrow \mathsf{Approve}(m, \sigma, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App}}, \mathsf{sk}_{\mathsf{App}})$.
5. *Verification: The* Verify *algorithm outputs a decision* $d \in \{\mathtt{true}, \mathtt{false}\}$*, verifying the signature* σ *for a message* m *w.r.t. the public keys* $\mathsf{pk}_{\mathsf{Sign}}$*,* $\mathsf{pk}_{\mathsf{App},1}$*, and* $\mathsf{pk}_{\mathsf{App},2}$*:* $d \leftarrow \mathsf{Verify}(m, \sigma, \mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\})$.

3.2 Correctness of 4EP Signature Schemes

As usual, we require the correctness properties to hold. In particular, we require that $\forall\lambda \in \mathbb{N}$, $\forall(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(1^\lambda)$, $\forall(\mathsf{pk}_{\mathsf{App},1}, \mathsf{sk}_{\mathsf{App},1}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$, $\forall(\mathsf{pk}_{\mathsf{App},2}, \mathsf{sk}_{\mathsf{App},2}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$, where $\mathsf{pk}_{\mathsf{App},1} \neq \mathsf{pk}_{\mathsf{App},2}$, $\forall\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\})$ we have $\mathtt{true} = \mathsf{Verify}(m, \sigma', \mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\})$, where $\sigma' \leftarrow \mathsf{Approve}(m, \mathsf{Approve}(m, \sigma, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},2}, \mathsf{sk}_{\mathsf{App},1}), \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},1}, \mathsf{sk}_{\mathsf{App},2})$, and also $\mathtt{true} = \mathsf{Verify}(m, \sigma'', \mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\})$, over all random coins used in any of the algorithms, where $\sigma'' \leftarrow \mathsf{Approve}(m, \mathsf{Approve}(m, \sigma, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},1}, \mathsf{sk}_{\mathsf{App},2}), \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},2}, \mathsf{sk}_{\mathsf{App},1})$. In other words, if both approvers approve the message m signed by the signer in any order, the signature must verify.

3.3 Security of 4EP-Signatures

Next, we introduce the required security guarantees these type of signatures must provide. In a nutshell, the main security guarantee we want to achieve is unforgeability, even against insiders. Only if *all* parties agree, the signature is considered valid. As we have three different entities, we need to consider all constellations. In the definitions, we ignore the case $\mathsf{pk}_{\mathsf{App},1} = \mathsf{pk}_{\mathsf{App},2}$, as this only happens with negligible probability.

3.4 Outsider Unforgeability

The first notion we introduce is outsider unforgeability. This definition requires that an adversary $\mathcal{A}$ not having any secret keys is not able to produce any validating signature σ^* corresponding to a message m^* it has never seen a signed, and fully approved, signature for.

Definition 6 (Outsider Unforgeability) *An* 4EPSIG *is outsider unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Outsider-Unforgeability}^{\mathsf{4EPSIG}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 3.*

3.5 Signer Unforgeability

The second notion we introduce is signer unforgeability. This definition requires that an adversary $\mathcal{A}$ able to generate the key pair for the signer is not able to produce any validating signature σ^* corresponding to a message m^* it has never seen a signed, and fully approved, signature for, if the approver public keys are generated honestly.

Definition 7 (Signer Unforgeability) *An* 4EPSIG *is signer unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that*

Experiment $\mathsf{Outsider-Unforgeability}^{\mathsf{4EPSIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}_{\mathsf{Sign}}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{App},1}, \mathsf{sk}_{\mathsf{App},1}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{App},2}, \mathsf{sk}_{\mathsf{App},2}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$
$\mathcal{Q}_1 = \mathcal{Q}_2 = \mathcal{Q}_3 \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{Sign}},\cdot),\mathsf{Approve}^1(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{App},1})}_{\mathsf{Approve}^2(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{App},2})}(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2})$
where oracle Sign on input $m_i, \mathsf{sk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\}$:
  let $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{(\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\}, m_i)\}$
  return σ
where oracle $\mathsf{Approve}^1$ on input $m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App},1}$:
  let $\sigma \leftarrow \mathsf{Approve}(m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App},1})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{(\mathsf{pk}_{\mathsf{Sign},i}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},i}\}, m_i)\}$
  return σ
where oracle $\mathsf{Approve}^2$ on input $m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App},2}$:
  let $\sigma \leftarrow \mathsf{Approve}(m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App},2})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q}_3 \leftarrow \mathcal{Q}_3 \cup \{(\mathsf{pk}_{\mathsf{Sign},i}, \{\mathsf{pk}_{\mathsf{App},i}, \mathsf{pk}_{\mathsf{App},2}\}, m_i)\}$
  return σ
return 1, if $\mathtt{true} = \mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}) \wedge$
  $((\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}, m^*) \notin \mathcal{Q}_1 \vee$
  $(\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}, m^*) \notin \mathcal{Q}_2 \vee$
  $(\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}, m^*) \notin \mathcal{Q}_3)$
return 0

Figure 3: Outsider Unforgeability

$\Pr[\mathsf{Signer-Unforgeability}^{\mathsf{4EPSIG}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 4.*

3.6 1Approver Unforgeability

The next notion we introduce is 1Approver unforgeability. This definition requires that an adversary $\mathcal{A}$ able to choose a single key pair for an approver, is not able to produce any validating signature σ^* for a message m^* it has never seen a signed, and fully approved, signature for.

Definition 8 (1Approver Unforgeability) *An* 4EPSIG *is 1Approver unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{1Approver-Unforgeability}^{\mathsf{4EPSIG}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 5.*

Experiment $\mathsf{Signer-Unforgeability}^{\mathsf{4EPSIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pk}_{\mathsf{App},1}, \mathsf{sk}_{\mathsf{App},1}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{App},2}, \mathsf{sk}_{\mathsf{App},2}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$
$\mathcal{Q}_1 = \mathcal{Q}_2 \leftarrow \emptyset$
$(\mathsf{pk}^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Approve}^1(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{App},1}),}_{\mathsf{Approve}^2(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{App},2})}(\{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\})$
where oracle $\mathsf{Approve}^1$ on input $m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App},1}$:
  let $\sigma \leftarrow \mathsf{Approve}(m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App},1})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{(\mathsf{pk}_{\mathsf{Sign},i}, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},i}\}, m_i)\}$
  return σ
where oracle $\mathsf{Approve}^2$ on input $m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App},2}$:
  let $\sigma \leftarrow \mathsf{Approve}(m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App},2})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{(\mathsf{pk}_{\mathsf{Sign},i}, \{\mathsf{pk}_{\mathsf{App},i}, \mathsf{pk}_{\mathsf{App},2}\}, m_i)\}$
  return σ
return 1, if $\mathtt{true} = \mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}^*, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}) \wedge$
  $((\mathsf{pk}^*, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}, m^*) \notin \mathcal{Q}_1 \vee$
  $(\mathsf{pk}^*, \{\mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}\}, m^*) \notin \mathcal{Q}_2)$
return 0

Figure 4: Signer Unforgeability

Experiment $\mathsf{1Approver-Unforgeability}^{\mathsf{4EPSIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}_{\mathsf{Sign}}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{App}}, \mathsf{sk}_{\mathsf{App}}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$
$\mathcal{Q}_1 = \mathcal{Q}_2 \leftarrow \emptyset$
$(\mathsf{pk}^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{Sign}},\cdot),\mathsf{Approve}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{App}})}(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App}})$
where oracle Sign on input $m_i, \mathsf{sk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\}$:
  let $\sigma \leftarrow \mathsf{Sign}(m_i, \mathsf{sk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{(\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\}, m_i)\}$
  return σ
where oracle $\mathsf{Approve}$ on input $m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App}}$:
  let $\sigma \leftarrow \mathsf{Approve}(m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App}})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{(\mathsf{pk}_{\mathsf{Sign},i}, \{\mathsf{pk}_{\mathsf{App},i}, \mathsf{pk}_{\mathsf{App}}\}, m_i)\}$
  return σ
return 1, if $\mathtt{true} = \mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}^*, \mathsf{pk}_{\mathsf{App}}\}) \wedge$
  $((\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}^*, \mathsf{pk}_{\mathsf{App}}\}, m^*) \notin \mathcal{Q}_1 \vee$
  $(\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}^*, \mathsf{pk}_{\mathsf{App}}\}, m^*) \notin \mathcal{Q}_2)$

Figure 5: 1Approver Unforgeability

$\mathbf{Experiment}\ 2\mathsf{Approver} - \mathsf{Unforgeability}_{\mathcal{A}}^{4\mathsf{EPSIG}}(\lambda)$
$(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}_{\mathsf{Sign}}(1^{\lambda})$
$\mathcal{Q} \leftarrow \emptyset$
$(\{\mathsf{pk}_1^*, \mathsf{pk}_2^*\}, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, \mathsf{sk}_{\mathsf{Sign}}, \cdot)}(\mathsf{pk}_{\mathsf{Sign}})$
where oracle Sign on input $m_i, \mathsf{sk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\}$:
  let $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}\}, m_i)\}$
  return σ
return 1, if $\mathtt{true} = \mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_1^*, \mathsf{pk}_2^*\}) \wedge$
  $(\mathsf{pk}_{\mathsf{Sign}}, \{\mathsf{pk}_1^*, \mathsf{pk}_2^*\}, m^*) \notin \mathcal{Q}$
return 0

Figure 6: 2Approver Unforgeability

3.7 2Approver Unforgeability

We also require that even if two approvers work together, they cannot generate any valid signature on a message m^* not endorsed by the signer. We call this 2Approver unforgeability. This definition requires that an adversary $\mathcal{A}$ is not able to generate both approvers' public keys, and a validating signature σ^* corresponding to a message m^* which has never been endorsed by an honest signer.

Definition 9 (2Approver Unforgeability) *An* 4EPSIG *is 2Approver unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[2\mathsf{Approver} - \mathsf{Unforgeability}_{\mathcal{A}}^{4\mathsf{EPSIG}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 6.*

3.8 Signer/Approver Unforgeability

The next notion we introduce is Signer/Approver unforgeability. This definition says that $\mathcal{A}$ is not able to produce any validating signature σ^* corresponding to a message m^* which was never approved by the honest approver, even it can choose the other public keys.

Definition 10 (Signer/Approver Unforgeability) *An* 4EPSIG *is Signer/Approver unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Signer/Approver} - \mathsf{Unforgeability}_{\mathcal{A}}^{4\mathsf{EPSIG}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 7.*

Definition 11 *We call an* 4EPSIG *secure, if it is correct, outsider unforgeable, signer unforgeable, 1Approver unforgeable, 2Approver unforgeable, and Signer/Approver unforgeable.*

Experiment $\mathsf{Signer/Approver - Unforgeability}_{\mathcal{A}}^{\mathsf{4EPSIG}}(\lambda)$
$(\mathsf{pk}_{\mathsf{App}}, \mathsf{sk}_{\mathsf{App}}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$(\mathsf{pk}_1^*, \mathsf{pk}_2^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Approve}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{App}})}(\mathsf{pk}_{\mathsf{App}})$
where oracle $\mathsf{Approve}$ on input $m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App}}$:
  let $\sigma \leftarrow \mathsf{Approve}(m_i, \sigma_i, \mathsf{pk}_{\mathsf{Sign},i}, \mathsf{pk}_{\mathsf{App},i}, \mathsf{sk}_{\mathsf{App}})$
  return $\perp$, if $\sigma = \perp$
  let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}_{\mathsf{Sign},i}, \{\mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App}}\}, m_i)\}$
  return σ
return 1, if $\mathtt{true} = \mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_1^*, \{\mathsf{pk}_2^*, \mathsf{pk}_{\mathsf{App}}\}) \wedge$
  $(\mathsf{pk}_1^*, \{\mathsf{pk}_2^*, \mathsf{pk}_{\mathsf{App}}\}, m^*) \notin \mathcal{Q}$
return 0

Figure 7: Signer/Approver Unforgeability

We stress that we define a new oracle for each approver.

3.9 Relations Between Security Properties

Due to the three entities involved, which are even more than in standard SSSs, we have to consider five different security properties. Obviously, as one would expect, some are stronger than others. We clarify this statement by proving the following theorems. Writing out the proofs also helps to recognize the emerging pattern by which our construction can easily be extended for more than two approvers.

Theorem 1 *Signer/Approver Unforgeability implies 1Approver Unforgeability.*

Proof 1 *We prove this theorem by a standard reduction. In particular, let $\mathcal{A}$ be the adversary breaking the 1Approver Unforgeability definition. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the Signer/Approver Unforgeability. $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{App}}$ from its own challenger. It then generates $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}_{\mathsf{Sign}}(1^\lambda)$. It passes $\mathsf{pk}_{\mathsf{App}}$, and $\mathsf{pk}_{\mathsf{Sign}}$ to $\mathcal{A}$ to initialize the adversary. The approve oracle can be simulated using the oracle provided. The signing oracle can be simulated honestly, as $\mathsf{sk}_{\mathsf{Sign}}$ is known. Eventually, $\mathcal{A}$ returns $(\mathsf{pk}^*, m^*, \sigma^*)$. By assumption, we know that m^* is fresh, $\mathcal{B}$ can return $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}^*, m^*, \sigma^*)$ as its own forgery attempt. The success probability of $\mathcal{B}$ equals the one of $\mathcal{A}$.*

Theorem 2 *2Approver Unforgeability implies 1Approver Unforgeability.*

Proof 2 *Let $\mathcal{A}$ be the adversary breaking the 1Approver Unforgeability definition. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to*

break the 2Approver Unforgeability. $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{Sign}}$ from its own challenger. It then generates $(\mathsf{pk}_{\mathsf{App}}, \mathsf{sk}_{\mathsf{App}}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$. It passes $\mathsf{pk}_{\mathsf{App}}$, and $\mathsf{pk}_{\mathsf{Sign}}$ to $\mathcal{A}$ to initialize the adversary. The signing oracle can be simulated using the signing oracle provided. The approve oracle can be simulated honestly, as $\mathsf{sk}_{\mathsf{App}}$ is known. Eventually, $\mathcal{A}$ returns $(\mathsf{pk}^*, m^*, \sigma^*)$. By assumption, we know that m^* is fresh, $\mathcal{B}$ can return $(\{\mathsf{pk}^*, \mathsf{pk}_{\mathsf{App}}\}, m^*, \sigma^*)$ as its own forgery attempt. The success probability of $\mathcal{B}$ equals the one of $\mathcal{A}$.

Theorem 3 *1Approver Unforgeability implies Outsider Unforgeability.*

Proof 3 *Let $\mathcal{A}$ be the adversary breaking the Outsider Unforgeability definition. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the 1Approver Unforgeability. $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{Sign}}$, and $\mathsf{pk}_{\mathsf{App},1}$ from its own challenger. It then generates $(\mathsf{pk}_{\mathsf{App},2}, \mathsf{sk}_{\mathsf{App},2}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$. It passes $\mathsf{pk}_{\mathsf{App},1}$, $\mathsf{pk}_{\mathsf{App},2}$, and $\mathsf{pk}_{\mathsf{Sign}}$ to $\mathcal{A}$ to initialize the adversary. The signing oracle can be simulated using the approve oracle provided. The approve oracle can be simulated honestly, as $\mathsf{sk}_{\mathsf{App}}$ is known. Eventually, $\mathcal{A}$ returns (m^*, σ^*). By assumption, we know that m^* is fresh, $\mathcal{B}$ can return $(\mathsf{pk}_{\mathsf{App},2}, m^*, \sigma^*)$ as its own forgery attempt. The success probability of $\mathcal{B}$ equals the one of $\mathcal{A}$.*

Theorem 4 *Signer/Approver Unforgeability implies Signer Unforgeability.*

Proof 4 *Let $\mathcal{A}$ be the adversary breaking the Outsider Unforgeability definition. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the Signer/Approver Unforgeability. $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{App},1}$ from its own challenger. It then generates $(\mathsf{pk}_{\mathsf{App},2}, \mathsf{sk}_{\mathsf{App},2}) \leftarrow \mathsf{KGen}_{\mathsf{App}}(1^\lambda)$. It passes $\mathsf{pk}_{\mathsf{App},1}$, $\mathsf{pk}_{\mathsf{App},2}$ to $\mathcal{A}$ to initialize the adversary. One approve oracle can be simulated honestly, as $\mathsf{sk}_{\mathsf{App},2}$ is known. The other approve oracle is the one provided to $\mathcal{B}$ itself. Eventually, $\mathcal{A}$ returns $(\mathsf{pk}^*, m^*, \sigma^*)$. By assumption, we know that m^* is fresh, $\mathcal{B}$ can return $(\mathsf{pk}^*, \mathsf{pk}_{\mathsf{App},2}, m^*, \sigma^*)$ as its own forgery attempt. The success probability of $\mathcal{B}$ equals the one of $\mathcal{A}$.*

Theorem 5 *Signer Unforgeability implies Outsider Unforgeability.*

Proof 5 *Let $\mathcal{A}$ be the adversary breaking the Outsider Unforgeability definition. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the 1Approver Unforgeability. $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{App},1}$, and $\mathsf{pk}_{\mathsf{App},2}$, from its own challenger. It then generates $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}_{\mathsf{Sign}}(1^\lambda)$. It passes $\mathsf{pk}_{\mathsf{App},1}$, $\mathsf{pk}_{\mathsf{App},2}$, and $\mathsf{pk}_{\mathsf{Sign}}$ to $\mathcal{A}$ to initialize the adversary. The approve oracles can be simulated using the oracles provided. The signing oracle can be simulated honestly, as $\mathsf{sk}_{\mathsf{App}}$ is known. Eventually, $\mathcal{A}$ returns (m^*, σ^*). By assumption, we know that m^* is fresh, $\mathcal{B}$ can return $(\mathsf{pk}_{\mathsf{Sign}}, m^*, \sigma^*)$ as its own forgery attempt. The success probability of $\mathcal{B}$ equals the one of $\mathcal{A}$.*

It is easy to see that our definitions, and the implications, can easily be extended for more than two approvers.

4 Construction of 4EP-Signatures

Next, we introduce our construction. The construction makes exclusive black-box use of SSS. The main idea is to use two SSS instances, signing the same message m. The first signature is sanitizable by the first approver, and the second signature by the second approver. Both signatures are generated by the entity generating the message's content m. In more detail, the initially unapproved content m is provided as non admissible, such that it cannot be changed by any approver. Each approver has to sanitize an admissible part, which was initially empty, into m and adjust the respective signature. A verifier then expects that both signatures point to the respective approver, as we require non-interactive public accountability.

Construction 1 (Secure 4EPSIG.) *We now construct* $\mathsf{4EPSIG} = (\mathsf{KGen_{sig}}, \mathsf{KGen_{App}}, \mathsf{Sign}, \mathsf{App}, \mathsf{Verify})$ *such that it is secure.*

$\mathsf{KGen_{sig}}$**.** *To generate the key pair for the signer, do the following steps.*

1. *Let* $(\mathsf{pk_{sig}}, \mathsf{sk_{sig}}) \leftarrow \mathsf{SSS.KGen_{sig}}(1^\lambda)$.
2. *Return* $(\mathsf{pk_{sig}}, \mathsf{sk_{sig}})$.

$\mathsf{KGen_{App}}$**.** *To generate the key pair for an approver, do the following steps.*

1. *Let* $(\mathsf{pk_{san}}, \mathsf{sk_{san}}) \leftarrow \mathsf{SSS.KGen_{san}}(1^\lambda)$.
2. *Return* $(\mathsf{pk_{san}}, \mathsf{sk_{san}})$.

Sign**.** *To generate a signature* σ*, on input of* m*,* $\mathsf{sk_{Sign}}$*,* $\{\mathsf{pk_{App,1}}, \mathsf{pk_{App,2}}\}$ *do the following steps. Note, we require canonical ordering of* $\{\mathsf{pk_{App,1}}, \mathsf{pk_{App,2}}\}$.

1. *If* $\mathsf{pk_{App,1}} = \mathsf{pk_{App,2}}$*, return* $\perp$.
2. *Set* $\mathrm{ADM} = (\{1\}, 5)$*, and* $m' = (\perp, m, \mathsf{pk_{App,1}}, \mathsf{pk_{App,2}}, \mathsf{pk_{Sign}})$.
3. *Let* $\sigma_1 \leftarrow \mathsf{SSS.Sign}(m', \mathsf{sk_{Sign}}, \mathsf{pk_{App,1}}, \mathrm{ADM})$.
4. *Let* $\sigma_2 \leftarrow \mathsf{SSS.Sign}(m', \mathsf{sk_{Sign}}, \mathsf{pk_{App,2}}, \mathrm{ADM})$.
5. *Return* (σ_1, σ_2).

Verify**.** *To verify a signature* $\sigma = (\sigma_1, \sigma_2)$*, on input* m*,* $\mathsf{pk_{Sign}}$*, and* $\{\mathsf{pk_{App,1}}, \mathsf{pk_{App,2}}\}$ *do:*

1. *Check that* $\mathrm{ADM}_1 = \mathrm{ADM}_2 = (\{1\}, 5)$*, where* ADM_1 *is taken from* σ_1*, and* ADM_2 *taken from* σ_2*. If not, return* $\perp$.

2. *Let* $m' = (m, m, \mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}, \mathsf{pk}_{\mathsf{Sign}})$.
3. *If* $\mathsf{San} = \mathsf{SSS.Judge}(m', \sigma_1, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},1}, \bot)$, *and* $\mathsf{San} = \mathsf{SSS.Judge}(m', \sigma_2, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},2}, \bot)$, *return* `true`.
4. *Return* `false`.

Note, if the "normal" verification fails, Judge *already outputs* Sig, *as we require non-interactive public accountability of the used* SSS.

Approve. *Let* $\mathsf{pk}'_{\mathsf{App}}$ *denote the public key corresponding to* $\mathsf{sk}_{\mathsf{App}}$. *Then, to approve a message* m, *on input of* $\sigma = (\sigma_1, \sigma_2)$, $\mathsf{pk}_{\mathsf{Sign}}$, $\mathsf{pk}_{\mathsf{App}}$, *and* $\mathsf{sk}_{\mathsf{App}}$, *do:*

1. *Return* $\bot$, *if* $\mathrm{ADM}_1 \neq \mathrm{ADM}_2 \neq (\{1\}, 5)$, *where* ADM_1 *is taken from* σ_1, *and* ADM_2 *taken from* σ_2, *or if* $\mathsf{pk}_{\mathsf{App}} = \mathsf{pk}'_{\mathsf{App}}$.
2. *If* $\mathsf{pk}'_{\mathsf{App}} \prec \mathsf{pk}_{\mathsf{App}}$, *let* $\mathsf{pk}_{\mathsf{App},1} \leftarrow \mathsf{pk}'_{\mathsf{App}}$, *and* $\mathsf{pk}_{\mathsf{App},2} \leftarrow \mathsf{pk}_{\mathsf{App}}$. *Otherwise, let* $\mathsf{pk}_{\mathsf{App},1} \leftarrow \mathsf{pk}_{\mathsf{App}}$, *and* $\mathsf{pk}_{\mathsf{App},2} \leftarrow \mathsf{pk}'_{\mathsf{App}}$.
3. *Let* $m' \leftarrow (\bot, m, \mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}, \mathsf{pk}_{\mathsf{Sign}})$.
4. *Let* $m'' \leftarrow (m, m, \mathsf{pk}_{\mathsf{App},1}, \mathsf{pk}_{\mathsf{App},2}, \mathsf{pk}_{\mathsf{Sign}})$.
5. *Let* $\mathrm{MOD} \leftarrow \{(1, m)\}$.
6. *Let* $d_{1,1} \leftarrow \mathsf{SSS.Verify}(m', \sigma_1, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},1})$.
7. *Let* $d_{1,2} \leftarrow \mathsf{SSS.Verify}(m'', \sigma_1, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},1})$.
8. *Let* $d_{2,1} \leftarrow \mathsf{SSS.Verify}(m', \sigma_2, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},2})$.
9. *Let* $d_{2,2} \leftarrow \mathsf{SSS.Verify}(m'', \sigma_2, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_{\mathsf{App},2})$.
10. *If* $\mathsf{pk}'_{\mathsf{App}} = \mathsf{pk}_{\mathsf{App},1}$, *do:*
    (a) *Return* $\bot$, *if* $d_{2,1} = d_{2,2} =$ `false`.
    (b) *Let* $(m'', \sigma') \leftarrow \mathsf{Sanit}(m', \mathrm{MOD}, \sigma_1, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{App}})$.
    (c) *Return* $\sigma = (\sigma', \sigma_2)$.
11. *If* $\mathsf{pk}'_{\mathsf{App}} = \mathsf{pk}_{\mathsf{App},2}$, *do:*
    (a) *Return* $\bot$, *if* $d_{1,1} = d_{1,2} =$ `false`.
    (b) *Let* $(m'', \sigma') \leftarrow \mathsf{Sanit}(m', \mathrm{MOD}, \sigma_2, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{App}})$.
    (c) *Return* (σ_1, σ').
12. *Return* $\bot$.

The proof of the following theorem is given in App. A.

Theorem 6 *If the underlying* SSS *is secure, then the above construction is secure.*

We stress that it is outside of the model whether the given public keys can be trusted. This can, e.g., be achieved by a standard PKI, or inter-organizational enforcement.

4.1 Extensions

Our construction paradigm can be enriched to achieve even more possibilities. We now present some of these alterations. We leave it as open work if these extensions are secure in a formal sense.

Multiple-Eyes Principle Sometimes, having two approvers for a given message m is not enough, especially if very important decisions are made. Our construction paradigm extends to this case in a straightforward way. The signer simply chooses more sanitizers, and adjusts the signed message m accordingly, i.e., adds more public keys, which need to be immutable.

Threshold Version We require that all approvers approve the message m before it is considered valid. A slight modification allows for a "t-out-of-n"-style signature scheme. Namely, the signer can also sign the information how many approvers are required before a signature becomes valid. Compared to standard threshold signature schemes, this also allows to see which party has actually approved a message m.

5 Conclusion and Future Work

We have shown how to enforce the four-eye principle by black-box access to sanitizable signatures. The underlying SSS is not required to fulfill all security requirements. We have then shown how to further alter our definitions, and the construction, to achieve additional goals such as a threshold version and more than two approvers. A still open problem is how to achieve Approver-Privacy, meaning that it is not clear which approver has approved a message, or if the other approver did not approve the message yet, and unlinkability.

Acknowledgments

A. Bilzhause was partly supported by BMBF grant agreement n° 01DH14022 (SECOR). H. C. Pöhls has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n° 644962 (PRISMACLOUD). K. Samelin was partly supported by ERC grant agreement n° 321310 (PERCY).

References

[1] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters. Computing on authenticated data. Cryptology ePrint Archive, Report 2011/096, 2011. http://eprint.iacr.org/.

[2] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures (extended abstract). In *EuroCrypt*, pages 591–606, 1998.

[3] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177, 2005.

[4] B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In *ICALP*, pages 524–535, 2000.

[5] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS*, pages 390–399, 2006.

[6] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC 2003*, pages 31–46, 2003.

[7] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT*, pages 416–432, 2003.

[8] C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of Sanitizable Signatures Revisited. In *Proc. of PKC 2009*, pages 317–336. Springer, 2009.

[9] C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Sanitizable signatures: How to partially delegate control for authenticated data. In *Proc. of BIOSIG*, volume 155 of *LNI*, pages 117–128. GI, 2009.

[10] C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of Sanitizable Signatures. In *PKC*, pages 444–461, 2010.

[11] C. Brzuska, H. C. Pöhls, and K. Samelin. Non-Interactive Public Accountability for Sanitizable Signatures. In *EuroPKI*, pages 178–193, 2012.

[12] C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In *EuroPKI*, pages 12–30, 2013.

[13] S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.

[14] S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AFRICACRYPT*, pages 35–52, 2012.

[15] S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.

[16] S. Canard and R. Lescuyer. Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In *ASIACCS*, pages 381–392, 2013.

[17] H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. Scope of security properties of sanitizable signatures revisited. In *ARES*, pages 188–197, 2013.

[18] H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. On the relation between redactable and sanitizable signature schemes. In *ESSoS*, pages 113–130, 2014.

[19] D. Demirel, D. Derler, C. Hanser, H. C. Pöhls, D. Slamanig, and G. Traverso. PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Technical report, H2020 Prismacloud, `www.prismacloud.eu`, 2015.

[20] D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig. A general framework for redactable signatures and new constructions. In *ICISC*, pages 3–19, 2015.

[21] D. Derler and D. Slamanig. Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In *ProvSec*, pages 455–474, 2015.

[22] V. Fehr and M. Fischlin. Sanitizable signcryption: Sanitization over encrypted data (full version). Cryptology ePrint Archive, Report 2015/765, 2015. `http://eprint.iacr.org/`.

[23] E. Ghosh, O. Ohrimenko, and R. Tamassia. Verifiable member and order queries on a list in zero-knowledge. *ePrint*, 632, 2014.

[24] J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In *InsCrypt*, volume 6584, pages 300–317, 2011.

[25] M. Klonowski and A. Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.

[26] S. Krenn, K. Samelin, and D. Sommer. Stronger security for sanitizable signatures. In *DPM*, pages 100–117, 2015.

[27] M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures for delegating signing operation. In *CCS '96*, pages 48–57, 1996.

[28] H. C. Pöhls, S. Peters, K. Samelin, J. Posegga, and H. de Meer. Malleable signatures for resource constrained platforms. In *WISTP*, pages 18–33, 2013.

[29] H. C. Pöhls and K. Samelin. Accountable redactable signatures. In *ARES*, pages 60–69, 2015.

[30] H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In *ACNS*, volume 6715 of *LNCS*, pages 166–182. Springer, 2011.

[31] V. Shoup. Practical threshold signatures. In *EuroCrypt*, pages 207–220, 2000.

[32] D. H. Yum, J. W. Seo, and P. J. Lee. Trapdoor sanitizable signatures made easy. In *ACNS*, pages 53–68, 2010.

A Proofs

Due to the given implications and separations, we only need to show that our construction is 2Approver unforgeable, and also Signer/Approver unforgeable. We prove each property on its own. Due to our choice of the given primitives, the reductions are tight, i.e., we have only constant reduction losses.

Theorem 7 *Our construction is 2Approver unforgeable.*

Proof 6 *In this case, we can reduce the security of our construction to immutability of the underlying* SSS*. In particular, we build an adversary* $\mathcal{B}$ *which uses* $\mathcal{A}$ *internally in a black-box way.* $\mathcal{B}$ *proceeds as follows. It receives* $\mathsf{pk}_{\mathsf{Sign}}$ *from its own challenger, and embeds it into* $\mathsf{pk}_{\mathsf{Sign}}$*. For every ith signing query,* $\mathcal{B}$ *uses its own signing oracle to generate two signatures on* $m' = (\bot, m, \mathsf{pk}_{\mathsf{App},1,i}, \mathsf{pk}_{\mathsf{App},2,i}, \mathsf{pk}_{\mathsf{Sign}})$ *(with the correct public key ordering) for* $\mathrm{ADM} = \{\{1\}, 5\}$*, and for* $\mathsf{pk}_{\mathsf{san}} = \mathsf{pk}_{\mathsf{App},1,i}$*, and the second signature for* $\mathsf{pk}_{\mathsf{App},2,i}$*. These signatures are given to* $\mathcal{A}$*. At some point,* $\mathcal{B}$ *returns its forgery attempt* $(\{\mathsf{pk}_1^*, \mathsf{pk}_2^*\}, m^*, \sigma^*)$*. As we already know that* $\sigma^* = (\sigma_1^*, \sigma_2^*)$*,* $\mathtt{true} = \mathsf{SSS.Verify}(m'^*, \sigma_1^*, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_1^*)$*, and also* $\mathtt{true} = \mathsf{SSS.Verify}(m'^*, \sigma_2^*, \mathsf{pk}_{\mathsf{Sign}}, \mathsf{pk}_2^*)$*, where* $m'^* = (m^*, m^*, \mathsf{pk}_1^*, \mathsf{pk}_2^*, \mathsf{pk}_{\mathsf{Sign}})$*, (or* $m'^* = (m^*, m^*, \mathsf{pk}_2^*, \mathsf{pk}_1^*, \mathsf{pk}_{\mathsf{Sign}})$*, depending on the ordering of the public keys),* $\mathcal{B}$ *can output* $(m'^*, \sigma_1^*, \mathsf{pk}_1^*)$ *or* $(m'^*, \sigma_2^*, \mathsf{pk}_2^*)$*, depending on which one is fresh (possibly even both), which can easily be deduced by looking at the signing queries. As* $\mathcal{B}$ *can perfectly simulate* $\mathcal{A}$*'s environment, and* m^* *is fresh by assumption (and thus also* m'^**), as at least one of the public keys is fresh (in the context with* m^**), the probability that* $\mathcal{B}$ *wins is the same as* $\mathcal{A}$*'s.*

Theorem 8 *Our construction is Signer/Approver unforgeable.*

Proof 7 *In this case, we only have to consider the case where $\mathcal{A}$ was able to generate a signature σ^* which verifies under the given public key, but was never sanitized, i.e., approved, but* Judge *decided* San*. Note that the message in question for the underlying* SSS *also contains the public keys. This case can be reduced to the non-interactive public-accountability of the used* SSS*. Namely, we can construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally. $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{san}}$ from its own challenger. $\mathsf{pk}_{\mathsf{Sign}}$ is discarded. It embeds $\mathsf{pk}_{\mathsf{san}}$ into $\mathsf{pk}_{\mathsf{App}}$. Every approving query is delegated to the sanitization oracle. Nothing else has to be simulated. At some point $\mathcal{A}$ returns $(\mathsf{pk}_1^*, \mathsf{pk}_2^*, m^*, \sigma^*)$. We already know, by assumption, that $(\mathsf{pk}_1^*, \{\mathsf{pk}_2^*, \mathsf{pk}_{\mathsf{App}}\}, m^*)$ is fresh. Thus, $\mathcal{B}$ can return $(\mathsf{pk}_2^*, m'^*, \sigma_1^*)$ or $(\mathsf{pk}_2^*, m''^*, \sigma_2^*)$ as its own forgery attempt, where $m'^* = (m^*, m^*, \mathsf{pk}_2^*, \mathsf{pk}_{\mathsf{App}}, \mathsf{pk}_1^*)$ or $m''^* = (m^*, m^*, \mathsf{pk}_1^*, \mathsf{pk}_2^*, \mathsf{pk}_{\mathsf{App}})$, depending on the ordering of the public keys. As $\mathcal{B}$ can perfectly simulate $\mathcal{A}$'s environment, $\mathcal{B}$'s success probability equals the one of $\mathcal{A}$.*

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, Sign, 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 (EUF-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

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

one 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 1A simple workflow examplefigure.1.1. For ease of presentation we label each outgoing edge with the respective type.

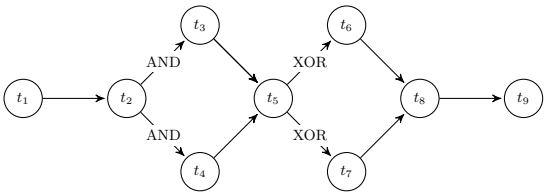

Fig. 1. A simple workflow example.

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 1A simple workflow examplefigure.1.1. 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 2Pattern for statically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. If OP = AND then $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = OR then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = XOR then $\hat{\sigma} \leftarrow \sigma_2$ or $\hat{\sigma} \leftarrow \sigma_3$figure.1.2 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

[4] When following the paradigm in Figure 2Pattern for statically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. If OP = AND then $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = OR then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = XOR then $\hat{\sigma} \leftarrow \sigma_2$ or $\hat{\sigma} \leftarrow \sigma_3$figure.1.2, the workflow manager is the delegator, whereas the agents are the delegatees. In contrast, following the paradigm in Figure 3Pattern 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 2Pattern for statically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. If OP = AND then $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = OR then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = XOR then $\hat{\sigma} \leftarrow \sigma_2$ or $\hat{\sigma} \leftarrow \sigma_3$figure.1.2)figure.1.3, agents act as both, delegatees and delegators, while only the first delegation is performed by the workflow manager.

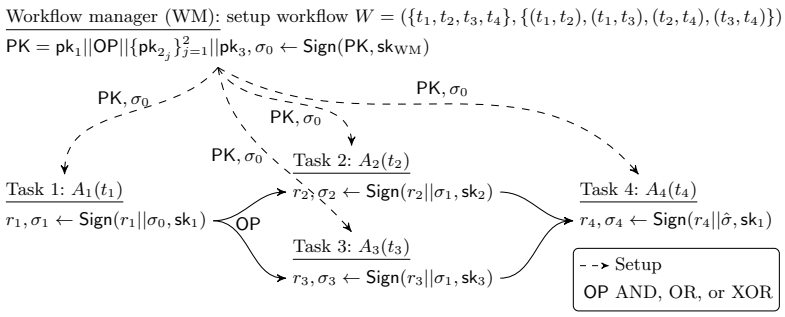

Fig. 2. Pattern for statically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. If OP = AND then $\hat{\sigma} \leftarrow \sigma_2 || \sigma_3$, if OP = OR then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow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

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 3Pattern 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 2Pattern for statically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. If OP = AND then $\hat{\sigma} \leftarrow \sigma_2 || \sigma_3$, if OP = OR then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow \sigma_2 || \sigma_3$, if OP = XOR then $\hat{\sigma} \leftarrow \sigma_2$ or $\hat{\sigma} \leftarrow \sigma_3$figure.1.2)figure.1.3 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.

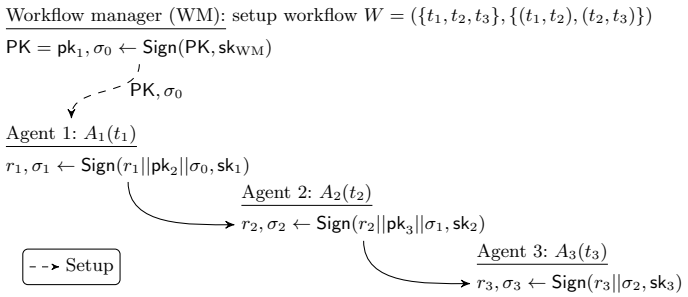

Fig. 3. Pattern for dynamically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. For simplicity, we omit split/join (cf. Figure 2Pattern for statically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. If OP = AND then $\hat{\sigma} \leftarrow \sigma_2 || \sigma_3$, if OP = OR then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow \sigma_2 || \sigma_3$, if OP = XOR then $\hat{\sigma} \leftarrow \sigma_2$ or $\hat{\sigma} \leftarrow \sigma_3$figure.1.2).

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

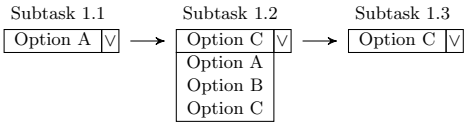

Fig. 4. A simple task report for a task $t_1 = (t_{1.1}, t_{1.2}, t_{1.3})$, composed of three multiple choice elements.

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.

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

if OP = OR then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = XOR then $\hat{\sigma} \leftarrow \sigma_2$ or $\hat{\sigma} \leftarrow \sigma_3$figure.1.2 and Figure 3Pattern 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 2Pattern for statically assigned workflows. The tuples (r_i, σ_i) denote the task reports corresponding to $A_i(t_i)$. If OP = AND then $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = OR then $\hat{\sigma} \leftarrow \sigma_2$, $\hat{\sigma} \leftarrow \sigma_3$, or $\hat{\sigma} \leftarrow \sigma_2||\sigma_3$, if OP = XOR then $\hat{\sigma} \leftarrow \sigma_2$ or $\hat{\sigma} \leftarrow \sigma_3$figure.1.2)figure.1.3. 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 (sk, 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, 9–11, 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 $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$.

$\mathsf{KeyGen_{san}}$: On input of a security parameter κ, this algorithm outputs a sanitizer key-pair $(\mathsf{sk_{san}}, \mathsf{pk_{san}})$.

Sign : On input of a message M, corresponding admissible modifications ADM, the keypair of the signer $(\mathsf{sk_{sig}}, \mathsf{pk_{sig}})$, as well as the verification key of the sanitizer $\mathsf{pk_{san}}$, 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 $\mathsf{pk_{sig}}$ and the secret key of the sanitizer $\mathsf{sk_{san}}$, 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 $\mathsf{pk_{sig}}$ and the sanitizer $\mathsf{pk_{san}}$, 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 $(\mathsf{sk_{sig}}, \mathsf{pk_{sig}})$ of the signer and the verification key of the sanitizer $\mathsf{pk_{san}}$, this algorithm outputs a proof π.

Judge : On input of a message-signature pair (M, σ), the verification keys of the signer $\mathsf{pk_{sig}}$ and the sanitizer $\mathsf{pk_{san}}$, 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 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.

[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.

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 ($\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify_T}, \mathsf{Inst}, \mathsf{Verify_I}$), which are introduced subsequently. Thereby, we assume that DSS signing keys for the originator ($\mathsf{sk_O}, \mathsf{pk_O}$) and the proxy ($\mathsf{sk_P}, \mathsf{pk_P}$) 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 $\mathsf{sk_O}$ and the verification key of the proxy $\mathsf{pk_P}$, 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 1Requirements covered by the respective instantiations. Legend: ✓ . . . supported, A . . . supported by ARSS [45], † . . . if scheme is transparenttable.1.1, we bring the various possible instantiations discussed above into the context of the previously defined requirements, where we exclude naive instantiations. Depending on the

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

used scheme, we can cover different subsets of previously 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

This is the full version of a paper which appears in Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016. Proceedings, Sara Foresti and Giuseppe Persiano Eds., Springer, LNCS 10052, pages 211–227, 10.1007/978-3-319-48965-0_13.

Signer-Anonymous Designated-Verifier Redactable Signatures for Cloud-Based Data Sharing

David Derler[1,‡], Stephan Krenn[2,‡,‖], and Daniel Slamanig[1,‡]

[1] IAIK, Graz University of Technology, Graz, Austria
{david.derler|daniel.slamanig}@tugraz.at
[2] AIT Austrian Institute of Technology GmbH, Vienna, Austria
stephan.krenn@ait.ac.at

Abstract. Redactable signature schemes allow to black out predefined parts of a signed message without affecting the validity of the signature, and are therefore an important building block in privacy-enhancing cryptography. However, a second look shows, that for many practical applications, they cannot be used in their vanilla form. On the one hand, already the identity of the signer may often reveal sensitive information to the receiver of a redacted message; on the other hand, if data leaks or is sold, everyone getting hold of (redacted versions of) a signed message will be convinced of its authenticity.

We overcome these issues by providing a definitional framework and practically efficient instantiations of so called *signer-anonymous designated-verifier redactable signatures* (AD-RS). As a byproduct we also obtain the first *group redactable signatures*, which may be of independent interest. AD-RS are motivated by a real world use-case in the field of health care and complement existing health information sharing platforms with additional important privacy features. Moreover, our results are not limited to the proposed application, but can also be directly applied to various other contexts such as notary authorities or e-government services.

1 Introduction

Digitalization of data and processes as well as the use of promising IT-trends such as cloud computing is prevalent, steadily increasing and meanwhile outreaches even sensitive fields such as the health care sector.[1] Given the sensitivity of the involved data and the high demands in data correctness and quality, the health care domain is a prime example for the beneficial application of cryptographic means such as encryption and digital signatures. This work is dedicated to the development of a cryptographically enhanced solution for a real world hospital, which is currently planning to complement its existing information sharing system for electronic patient data with additional privacy features. The overall idea

‡ Supported by EU H2020 project PRISMACLOUD, grant agreement n°644962.
‖ Supported by EU H2020 project CREDENTIAL, grant agreement n°653454.
[1] See e.g., www.healthcaredive.com/news/407746/

of the system is to grant patients access to all their medical records via a cloud-based platform. The patients are then able to use this as a central hub to distribute their documents to different stakeholders, e.g., to request reimbursement by the insurance, or to forward (parts of) the documents to the family doctor for further treatment. While means for access control and data confidentiality are already in place, the system should be complemented by strong authenticity guarantees. At the same time a high degree of privacy should be maintained, i.e., by allowing the patients, on a fine-granular basis, to decide which parts of which document should be visible to which party. For instance, the family doctor might *not* need to learn the precise costs of a treatment; similarly a medical research laboratory should *not* learn the patients' identities.

From a research point of view, one motivation behind this work is to show how rather complex real world scenarios with conflicting interests and strong security and privacy requirements can be elegantly and securely realized by means of rigorous cryptographic design and analysis. More importantly, we can indeed come up with provably secure and practical solutions being well suited for real world use. Now, we discuss the motivation for our design.

Redactable Signatures. A trivial solution for the above problem would be to let the hospital cloud create a fresh signature on the information to be revealed every time the user wishes to forward authentic subsets of a document to other parties. However, this is not satisfactory as it would require strong trust assumptions into the cloud: one could not efficiently guarantee that the signed data has not been altered over time by the cloud or by a malicious intruder. It is therefore preferable to use *redactable signatures* (RS). These are signature schemes that allow to black out (redact) predefined parts of a signed message while preserving the validity of the signature, thereby guaranteeing the authenticity of the redacted message. That is, it is not necessary to let the cloud attest the authenticity of the forwarded data, as the signature on the redacted document can be extracted from the doctor's signature on the original document without requiring the doctor's secret signing key or further interaction with the doctor.

Designated Verifiers. Unfortunately, using redactable signatures in their vanilla form in our scenario would lead to severe privacy problems, i.e., everyone getting hold of a signed document would be convinced of its authenticity. In such a case, for instance, an employer who gets hold of a signed health record of an employee, might reliably learn the employee's disease, who, in further consequence, might get dismissed. What is therefore needed is a *designated verifier* for each redacted version of a document. That is, when redacting a document, the patient should be able to define the intended receiver. Then, while everybody can check the validity of a leaked document, *only* the designated verifier is convinced about its authenticity. This can be achieved by constructing the schemes in a way that the designated verifier can fake indistinguishable signatures on its own. Moreover, the public verifiability property might as well be a motivation for designated verifiers to not leak/sell documents, as this reduces the circle of possible suspects to the data owner and the designated verifier.

Group Signatures. Another problem of RS is that they only support a single signer. However, a hospital potentially employing hundreds of doctors will not use a single signing key that is shared by all its employees. By doing so, the identity of the signing doctor could not be revealed in case of a dispute, e.g., after a malpractice. However, using different keys for different doctors poses a privacy risk again. For instance, if the document was signed using an oncologist's key, one could infer sensitive information about the disease—even though the diagnosis was blacked out. What is therefore needed are features known from *group signatures*, where towards the verifier the doctor's identity remains hidden within the set of doctors in the hospital, while re-identification is still possible by a dedicated entity.

Contribution. The properties we need for our scenario are contributed by three distinct cryptographic concepts and what we actually need can be considered as a *signer-anonymous designated-verifier redactable signature scheme*. However, while a lot of existing work studies the different concepts in isolation, there is no work which aims at combining them in a way to profit from a combination of their individual properties. Trying to obtain this by simply combining them in an ad-hoc fashion, however, is dangerous. It is well known that the ad-hoc combination of cryptographic primitives to larger systems is often problematic (as subtle issues often remain hidden when omitting formal analysis) and security breaches resulting from such approaches are often seen in practice. Unlike following such an ad-hoc approach, we follow a rigorous approach and formally model what is required by the use-case, introduce a comprehensive security model and propose two (semi-)black-box constructions that are provably secure within our model. While such a (semi-)black-box construction is naturally interesting from a theoretical point of view, our second construction is also entirely practical and thus also well suited to be used within the planned system. Finally, as a contribution which may be of independent interest, we also obtain the first *group redactable signatures* as a byproduct of our definitional framework.

Technical Overview. Our constructions provably achieve the desired functionality by means of a two-tier signature approach: a message is signed using a freshly generated RS key pair where the corresponding public key of this "one-time RS" is certified using a group signature. For the designated verifier feature, we follow two different approaches. Firstly, we follow the naïve approach and use a disjunctive non-interactive proof of knowledge which either demonstrates knowledge of a valid RS signature on the message, *or* it demonstrates knowledge of a valid signature of the designated verifier on the same message. While this approach is very generic, its efficiency largely depends on the complexity to prove knowledge of an RS signature. To this end, we exploit key-homomorphic signatures, which we introduce and which seem to be of independent interest. In particular, we use the observation that a large class of RS can easily be turned into RS admitting the required key-homomorphism, to obtain a practical construction. More precisely, besides conventional group signatures and conventional redactable signatures, our approach only requires to prove a single statement demonstrating knowledge of the relation between two RS keys *or* demonstrating knowledge of

the designated verifier's secret key. For instance, in the discrete logarithm setting when instantiating this proof using Fiat-Shamir transformed [FS86] Σ-protocols, they are highly efficient as they only require two group exponentiations.

Related Work. Redactable signature schemes have been independently introduced in [JMSW02] and [SBZ01]. Although such schemes suffer from the aforementioned problems, we can use them as an important building block. In particular, we will rely on the general framework for such signatures as presented in [DPSS15]. Besides that, redactable signatures with an unlinkability property have been introduced in [CDHK15, PS15].[2] Unfortunately, apart from lacking practical efficiency, even unlinkable redactable signatures are not useful to achieve the desired designated verifier functionality. There is a large body of work on signatures with designated verifiers, which are discussed subsequently. However, none of the approaches considers selective disclosure via redaction or a group signing feature.

In designated verifier (DV) signatures (or proofs) [JSI96], a signature produced by a signer can only be validated by a single user who is designated by the signer during the signing process (cf. [LWB05] for a refined security model). Designation can only be performed by the signer and verification requires the designated verifier's secret. Thus, this concept is not directly applicable to our setting. In [JSI96] also the by now well known "*OR trick*" was introduced as a DV construction paradigm.

Undeniable signatures [CA89] are signatures that can not be verified without the signer's cooperation and the signer can either prove that a signature is valid or invalid. This is not suitable for us as this is an interactive process.

Designated confirmer signatures [Cha94] introduce a third entity besides the signer and the verifier called designated confirmer. This party, given a signature, has the ability to privately verify it as well as to convince anyone of its validity or invalidity. Additionally, the designated confirmer can convert a designated confirmer signature into an ordinary signature that is then publicly verifiable. This is not suitable for our scenario, as it is exactly the opposite of what we require, i.e., here the signature for the confirmer is not publicly verifiable, but the confirmer can always output publicly verifiable versions of this signature.

Another concept, which is closer to the designation functionality that we require, are universal designated verifier (UDV) signatures introduced in [SBWP03]. They are similar to designated verifier signatures, but universal in the sense that any party who is given a publicly verifiable signature from the signer can designate the signature to any designated verifier by using the verifiers public key. Then, the designated verifier can verify that the message was signed by the signer, but is unable to convince anyone else of this fact. Like with ordinary DV signatures, UDV signatures also require the designated verifier's secret key for verification. There are some generic results for UDV signatures. In [Ver06] it was shown how to convert various pairing-based signature schemes into UDV signatures. In [SS08] it was shown how to convert a large class of signature schemes

[2] Similar to the related concept of unlinkable sanitizable signatures [BFLS10, BPS13, FKM+16, LZCS16].

into UDV signatures. Some ideas in our second construction are conceptually related to this generic approach. However, as we only require to prove relations among public keys, our approach is more tailored to efficiency.

2 Preliminaries

We denote algorithms by sans-serif letters, e.g., A, B. All algorithms are assumed to return a special symbol $\perp$ on error. By $y \leftarrow \mathsf{A}(x)$, we denote that y is assigned the output of the potentially probabilistic algorithm A on input x and fresh random coins. Similarly, $y \xleftarrow{R} S$ means that y was sampled uniformly at random from a set S. We let $[n] := \{1, \ldots, n\}$. We write $\Pr[\Omega : \mathcal{E}]$ to denote the probability of an event $\mathcal{E}$ over the probability space Ω. We use $\mathcal{C}$ to denote challengers of security experiments, and $\mathcal{C}_\kappa$ to make the security parameter explicit.

A function $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is called negligible, iff it vanishes faster than every inverse polynomial, i.e., $\forall\, k : \exists\, n_k : \forall\, n > n_k : \varepsilon(n) < n^{-k}$.

Followingly, we recap required cryptographic building blocks. Due to space constraints we omit formal definitions for well known primitives such as a digital signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ and a (non-interactive) proof system $\Pi = (\mathsf{Setup}, \mathsf{Proof}, \mathsf{Verify})$ here, and present them in Appendix A.

Redactable Signatures. Below, we recall the generalized model for redactable signatures from [DPSS15], which builds up on [BBD+10]. As done in [DPSS15], we do not make the structure of the message explicit. That is, we assume that the message m to be signed is some arbitrarily structured data. We use ADM to denote a data structure encoding the admissible redactions of some messeage m and we use MOD to denote a data structure containing modification instructions for some message. We use $\mathring{\mathsf{m}} \overset{\mathsf{ADM}}{\preceq} \mathsf{m}$ to denote that a message $\mathring{\mathsf{m}}$ is derivable from a message m under ADM and $\mathring{\mathsf{m}} \xleftarrow{\mathsf{MOD}} \mathsf{m}$ to denote that $\mathring{\mathsf{m}}$ is obtained by applying MOD to m. Likewise, we use $\mathring{\mathsf{ADM}} \xleftarrow{\mathsf{MOD}} \mathsf{ADM}$ to denote the derivation of $\mathring{\mathsf{ADM}}$ from ADM with respect to MOD. We use $\mathsf{ADM} \preceq \mathsf{m}$ to denote that ADM matches m, and $\mathsf{MOD} \preceq \mathsf{ADM}$ to denote that MOD matches ADM.

Definition 1. *An* RS *is a tuple* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact})$ *of PPT algorithms, which are defined as follows:*

$\mathsf{KeyGen}(1^\kappa)$: *Takes a security parameter* κ *as input and outputs a keypair* $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{Sign}(\mathsf{sk}, \mathsf{m}, \mathsf{ADM})$: *Takes a secret key* sk*, a message* m *and admissible modifications* ADM *as input, and outputs a message-signature pair* (m, σ) *together with some auxiliary redaction information* RED.[3]

$\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \sigma)$: *Takes a public key* pk*, a message* m*, and a signature* σ *as input, and outputs a bit* b.

$\mathsf{Redact}(\mathsf{pk}, \mathsf{m}, \sigma, \mathsf{MOD}, \mathsf{RED})$: *Takes a public key* pk*, a message* m*, a valid signature* σ*, modification instructions* MOD*, and auxiliary redaction information* RED *as input. It returns a redacted message-signature pair* $(\mathring{\mathsf{m}}, \mathring{\sigma})$ *and an updated auxiliary redaction information* $\mathring{\mathsf{RED}}$.

[3] As it is common for RS, we assume that ADM can always be recovered from (m, σ).

While we omit correctness, we recall the remaining RS security definitions below.

Definition 2 (Unforgeability). *An* RS *is unforgeable, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa),\\ (\mathsf{m}^\star,\sigma^\star) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot,\cdot)}(\mathsf{pk})\end{array} : \begin{array}{r}\mathsf{Verify}(\mathsf{pk},\mathsf{m}^\star,\sigma^\star)=1 \ \wedge\\ \nexists\ (\mathsf{m},\mathsf{ADM}) \in \mathcal{Q}^{\mathsf{Sign}} : \mathsf{m}^\star \overset{\mathsf{ADM}}{\preceq} \mathsf{m}\end{array}\right] \leq \varepsilon(\kappa),$$

where the environment keeps track of the queries to the signing oracle via $\mathcal{Q}^{\mathsf{Sign}}$.

Definition 3 (Privacy). *An* RS *is private, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa),\ b \xleftarrow{R} \{0,1\},\\ \mathcal{O} \leftarrow \{\mathsf{Sign}(\mathsf{sk},\cdot,\cdot), \mathsf{LoRRedact}(\mathsf{sk},\mathsf{pk},\cdot,\cdot,b)\},\\ b^\star \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk})\end{array} : b = b^\star\right] \leq 1/2 + \varepsilon(\kappa),$$

where LoRRedact *is defined as follows:*

$\mathsf{LoRRedact}(\mathsf{sk},\mathsf{pk},(\mathsf{m}_0,\mathsf{ADM}_0,\mathsf{MOD}_0),(\mathsf{m}_1,\mathsf{ADM}_1,\mathsf{MOD}_1),b)$*:*

1: *Compute* $((\mathsf{m}_c,\sigma_c),\mathsf{RED}_c) \leftarrow \mathsf{Sign}(\mathsf{sk},\mathsf{m}_c,\mathsf{ADM}_c)$ *for* $c \in \{0,1\}$.
2: *Let* $((\mathring{\mathsf{m}}_c,\mathring{\sigma}_c),\mathring{\mathsf{RED}}_c) \leftarrow \mathsf{Redact}(\mathsf{pk},\sigma_c,\mathsf{m}_c,\mathsf{MOD}_c,\mathsf{RED}_c)$ *for* $c \in \{0,1\}$.
3: *If* $\mathring{\mathsf{m}}_0 \neq \mathring{\mathsf{m}}_1 \ \vee\ \mathring{\mathsf{ADM}}_0 \neq \mathring{\mathsf{ADM}}_1$, *return* $\perp$.
4: *Return* $(\mathring{\mathsf{m}}_b,\mathring{\sigma}_b)$.

Here, the admissible modifications $\mathring{\mathsf{ADM}}_0$ *and* $\mathring{\mathsf{ADM}}_1$ *corresponding to the redacted messages are implicitly defined by (and recoverable from) the tuples* $(\mathring{\mathsf{m}}_0,\mathring{\sigma}_0)$ *and* $(\mathring{\mathsf{m}}_1,\mathring{\sigma}_1)$ *and the oracle returns* $\perp$ *if any of the algorithms returns* $\perp$.

We call an RS *secure*, if it is correct, unforgeable, and private.

Group Signatures. Subsequently, we recall the established model for static group signatures from [BMW03]. Again, we slightly adapt the notation to ours.

Definition 4. *A group signature scheme* GS *is a tuple* (KeyGen, Sign, Verify, Open) *of PPT algorithms which are defined as follows:*

$\mathsf{KeyGen}(1^\kappa,n)$: *Takes a security parameter* κ *and the group size* n *as input. It generates and outputs a group verification key* gpk*, a group opening key* gok*, as well as a list of group signing keys* $\mathsf{gsk} = \{\mathsf{gsk}_i\}_{i\in[n]}$.

$\mathsf{Sign}(\mathsf{gsk}_i,\mathsf{m})$: *Takes a group signing key* gsk_i *and a message* m *as input and outputs a signature* σ.

$\mathsf{Verify}(\mathsf{gpk},\mathsf{m},\sigma)$: *Takes a group verification key* gpk*, a message* m *and a signature* σ *as input, and outputs a bit* b.

$\mathsf{Open}(\mathsf{gok},\mathsf{m},\sigma)$: *Takes a group opening key* gok*, a message* m *and a signature* σ *as input, and outputs an identity* i.

The GS security properties are formally defined as follows (we omit correctness).

Definition 5 (Anonymity). *A* GS *is anonymous, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l|r} (\mathsf{gpk},\mathsf{gok},\mathsf{gsk}) \leftarrow \mathsf{KeyGen}(1^\kappa, n), & \\ b \xleftarrow{R} \{0,1\},\ \mathcal{O} \leftarrow \{\mathsf{Open}(\mathsf{gok},\cdot,\cdot)\}, & b = b^\star \wedge \\ (i_0^\star, i_1^\star, \mathsf{m}^\star, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{gpk},\mathsf{gsk}), & (\mathsf{m}^\star,\sigma) \notin \mathcal{Q}_2^{\mathsf{Open}} \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{gsk}_{i_b^\star}, \mathsf{m}^\star),\ b^\star \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma,\mathsf{st}) & \end{array}\right] \leq \varepsilon(\kappa),$$

where $\mathcal{A}$ *runs in two stages and* $\mathcal{Q}_2^{\mathsf{Open}}$ *records the* Open *queries in stage two.*

Definition 6 (Traceability). *A* GS *is traceable, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l|r} (\mathsf{gpk},\mathsf{gok},\mathsf{gsk}) \leftarrow \mathsf{KeyGen}(1^\kappa, n), & \mathsf{Verify}(\mathsf{gpk},\mathsf{m}^\star,\sigma^\star) = 1 \wedge \\ \mathcal{O} \leftarrow \{\mathsf{Sig}(\cdot,\cdot), \mathsf{Key}(\cdot)\}, & (i = \perp \ \vee \ (i \notin \mathcal{Q}^{\mathsf{Key}} \wedge \\ (\mathsf{m}^\star,\sigma^\star) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{gpk},\mathsf{gok}), & (i,\mathsf{m}^\star) \notin \mathcal{Q}^{\mathsf{Sig}})) \\ i \leftarrow \mathsf{Open}(\mathsf{gok},\mathsf{m}^\star,\sigma^\star) & \end{array}\right] \leq \varepsilon(\kappa),$$

where $\mathsf{Sig}(i,\mathsf{m})$ *returns* $\mathsf{Sign}(\mathsf{gsk}_i,\mathsf{m})$, $\mathsf{Key}(i)$ *returns* gsk_i, *and* $\mathcal{Q}^{\mathsf{Sig}}$ *and* $\mathcal{Q}^{\mathsf{Key}}$ *record the queries to the signing and key oracle respectively.*

We call a GS *secure*, if it is correct, anonymous and traceable.

3 Security Model

Now we formally define signer-anonymous designated-verifier redactable signature schemes (AD-RS). To obtain the most general result, we follow [DPSS15] and do not make the structure of the messages to be signed explicit. Inspired by [MPV09], we view signatures output by Sign as being of the form $\sigma = (\underline{\sigma}, \overline{\sigma})$. That is, signatures are composed of a public signature component $\underline{\sigma}$ and a private signature component $\overline{\sigma}$, where $\underline{\sigma}$ may also be empty. For the sake of simple presentation we model our system for static groups, since an extension to dynamic groups [BSZ05] is straight forward.

Definition 7 (AD-RS). *An* AD-RS *is a tuple* (Setup, DVGen, Sign, GVerify, Open, Redact, Verify, Sim) *of PPT algorithms, which are defined as follows.*

$\mathsf{Setup}(1^\kappa, n)$: *Takes a security parameter* κ *and the group size* n *as input. It generates and outputs a group public key* gpk, *a group opening key* gok, *and a list of group signing keys* $\mathsf{gsk} = \{\mathsf{gsk}_i\}_{i\in[n]}$.

$\mathsf{DVGen}(1^\kappa)$: *Takes a security parameter* κ *as input and outputs a designated verifier key pair* $(\mathsf{vsk}_j, \mathsf{vpk}_j)$.

$\mathsf{Sign}(\mathsf{gsk}_i, \mathsf{m}, \mathsf{ADM})$: *Takes a group signing key* gsk_i, *a message* m, *and admissible modifications* ADM *as input, and outputs a signature* σ.

$\mathsf{GVerify}(\mathsf{gpk}, \mathsf{m}, \sigma)$: *Takes a group public key* gpk, *a message* m, *and a signature* σ *as input, and outputs a bit* b.

$\mathsf{Open}(\mathsf{gok}, \mathsf{m}, \sigma)$: *Takes a group opening key* gok*, a message* m*, and a valid signature* σ *as input, and outputs an identity* i*.*

$\mathsf{Redact}(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \sigma, \mathsf{MOD})$: *Takes a group public key* gpk*, a designated-verifier public key* vpk_j*, a message* m*, a valid signature* σ*, and modification instructions* MOD *as input, and returns a designated-verifier message-signature pair* $(\mathring{\mathsf{m}}, \rho)$*.*

$\mathsf{Verify}(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \rho)$: *Takes a group public key* gpk*, a designated-verifier public key* vpk_j*, a message* m*, and a designated-verifier signature* ρ*. It returns a bit* b*.*

$\mathsf{Sim}(\mathsf{gpk}, \mathsf{vsk}_j, \mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \underline{\sigma})$**:** *Takes a group public key* gpk*, a designated-verifier secret key* vsk_j*, a message* m*, admissible modifications* ADM*, modification instructions* MOD*, and a valid public signature component* $\underline{\sigma}$ *as input and outputs a designated-verifier message signature pair* $(\mathring{\mathsf{m}}, \rho)$*.*

Oracles. We base our security notions on the following oracles and assume that $(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk})$ generated in the experiments are implicitly available to them. The environment stores a list $\mathtt{DVK}$ of designated-verifier key pairs, and a set of public signature components $\mathtt{SIG}$. Each list entry and each set is initially set to $\perp$.

$\mathsf{Key}(i)$: This oracle returns gsk_i.

$\mathsf{DVGen}(j)$: If $\mathtt{DVK}[j] \neq \perp$ this oracle returns $\perp$. Otherwise, it runs $(\mathsf{vsk}_j, \mathsf{vpk}_j) \leftarrow \mathsf{DVGen}(1^\kappa)$, sets $\mathtt{DVK}[j] \leftarrow (\mathsf{vsk}_j, \mathsf{vpk}_j)$, and returns vpk_j.

$\mathsf{DVKey}(j)$: This oracle returns vsk_j.

$\mathsf{Sig}(i, \mathsf{m}, \mathsf{ADM})$: This oracle runs $\sigma = (\underline{\sigma}, \overline{\sigma}) \leftarrow \mathsf{Sign}(\mathsf{gsk}_i, \mathsf{m}, \mathsf{ADM})$, sets $\mathtt{SIG} \leftarrow \mathtt{SIG} \cup \{\underline{\sigma}\}$ and returns σ.

$\mathsf{Open}(\mathsf{m}, \sigma)$: This oracle runs $i \leftarrow \mathsf{Open}(\mathsf{gok}, \mathsf{m}, \sigma)$ and returns i.

$\mathsf{Sim}(j, \mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \underline{\sigma})$: If $\underline{\sigma} \notin \mathtt{SIG}$, this oracle returns $\perp$. Otherwise, it runs $(\mathring{\mathsf{m}}, \rho) \leftarrow \mathsf{Sim}(\mathsf{gpk}, \mathsf{vsk}_j, \mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \underline{\sigma})$ and returns $(\mathring{\mathsf{m}}, \rho)$.

$\mathsf{RoS}(b, j, \mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \sigma)$: If $b = 0$, this oracle runs $(\mathring{\mathsf{m}}, \rho) \leftarrow \mathsf{Redact}(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \sigma, \mathsf{MOD})$ and returns $(\mathring{\mathsf{m}}, \rho)$. Otherwise, it uses the Sim oracle to obtain $(\mathring{\mathsf{m}}, \rho) \leftarrow \mathsf{Sim}(j, \mathsf{m}, \mathsf{ADM}, \mathsf{MOD}, \underline{\sigma})$ and returns $(\mathring{\mathsf{m}}, \rho)$.

$\mathsf{Ch}(i, j, (\mathsf{m}_0, \mathsf{ADM}_0, \mathsf{MOD}_0), (\mathsf{m}_1, \mathsf{ADM}_1, \mathsf{MOD}_1), b)$: This oracle runs $\sigma_c \leftarrow \mathsf{Sign}(\mathsf{gsk}_i, \mathsf{m}_c, \mathsf{ADM}_c)$, $(\mathring{\mathsf{m}}_c, \rho_c) \leftarrow \mathsf{Redact}(\mathsf{vpk}_j, \mathsf{m}_c, \sigma_c, \mathsf{MOD}_c)$, for $c \in \{0, 1\}$. If $\mathring{\mathsf{m}}_0 \neq \mathring{\mathsf{m}}_1 \ \vee \ \mathring{\mathsf{ADM}}_0 \neq \mathring{\mathsf{ADM}}_1$, it returns $\perp$ and $(\mathring{\mathsf{m}}_b, \underline{\sigma}_b, \rho_b)$ otherwise.[4]

The environment stores the oracle queries in lists. In analogy to the oracle labels, we use $\mathcal{Q}^{\mathsf{Key}}, \mathcal{Q}^{\mathsf{DVGen}}, \mathcal{Q}^{\mathsf{DVKey}}, \mathcal{Q}^{\mathsf{Sig}}, \mathcal{Q}^{\mathsf{Open}}, \mathcal{Q}^{\mathsf{Sim}}, \mathcal{Q}^{\mathsf{RoS}}$, and $\mathcal{Q}^{\mathsf{Ch}}$ to denote them.

Security Notions. We require AD-RS to be correct, group unforgeable, designated-verifier unforgeable, simulatable, signer anonymous, and private.

Correctness guarantees that all honestly computed signatures verify correctly.

Formally, we require that for all $\kappa \in \mathbb{N}$, for all $n \in \mathbb{N}$, for all $(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\kappa, n)$, for all $(\mathsf{vsk}_j, \mathsf{vpk}_j) \leftarrow \mathsf{DVGen}(1^\kappa)$, for all $(\mathsf{vsk}_\ell, \mathsf{vpk}_\ell) \leftarrow \mathsf{DVGen}(1^\kappa)$, for all $(\mathsf{m}, \mathsf{ADM}, \mathsf{MOD})$ where $\mathsf{MOD} \preceq \mathsf{ADM} \wedge \mathsf{ADM} \preceq \mathsf{m}$, for all $(\mathsf{m}', \mathsf{ADM}', \mathsf{MOD}')$ where $\mathsf{MOD}' \preceq \mathsf{ADM}' \ \wedge \ \mathsf{ADM}' \preceq \mathsf{m}'$ for all $i \in [n]$, for all $\sigma = (\underline{\sigma}, \overline{\sigma}) \leftarrow \mathsf{Sign}(\mathsf{gsk}_i, \mathsf{m}, \mathsf{ADM})$,

[4] Here $\mathring{\mathsf{ADM}}_0$ and $\mathring{\mathsf{ADM}}_1$ are derived from ADM_0 and ADM_1 with respect to MOD_0 and MOD_1.

for all $u \leftarrow \mathsf{Open}(\mathsf{gok}, \mathsf{m}, \sigma)$, for all $(\mathring{\mathsf{m}}, \rho) \leftarrow \mathsf{Redact}(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \sigma, \mathsf{MOD})$, for all $(\mathring{\mathsf{m}}', \rho') \leftarrow \mathsf{Sim}(\mathsf{gpk}, \mathsf{vsk}_\ell, \mathsf{m}', \mathsf{ADM}', \mathsf{MOD}', \underline{\sigma})$, it holds with overwhelming probability that $\mathsf{GVerify}(\mathsf{gpk}, \mathsf{m}, \sigma) = 1 \ \wedge \ i = u \ \wedge \ \mathsf{Verify}(\mathsf{gpk}, \mathsf{vpk}_j, \mathring{\mathsf{m}}, \rho) = 1 \ \wedge \ \mathsf{Verify}(\mathsf{gpk}, \mathsf{vpk}_\ell, \mathring{\mathsf{m}}', \rho') = 1$ and that $\mathring{\mathsf{m}} \overset{\text{MOD}}{\longleftarrow} \mathsf{m} \ \wedge \ \mathring{\mathsf{m}}' \overset{\text{MOD}'}{\longleftarrow} \mathsf{m}'$.

Group unforgeability captures the intuition that the only way of obtaining valid signatures on messages is by applying "allowed" modifications to messages which were initially signed by a group member. Moreover, this property guarantees that every valid signature can be linked to the original signer by some authority.

Technically, the definition captures the traceability property of group signatures while simultaneously taking the malleability of RS into account.

Definition 8. *An* AD-RS *is group unforgeable, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\kappa, n), \\ \mathcal{O} \leftarrow \{\mathsf{Sig}(\cdot,\cdot,\cdot), \mathsf{Key}(\cdot)\}, \\ (\mathsf{m}^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{gpk}, \mathsf{gok}), \\ u \leftarrow \mathsf{Open}(\mathsf{gok}, \mathsf{m}^\star, \sigma^\star)\end{array} : \begin{array}{r}\mathsf{GVerify}(\mathsf{gpk}, \mathsf{m}^\star, \sigma^\star) = 1 \ \wedge \\ (u = \bot \ \vee \ (u \notin \mathcal{Q}^{\mathsf{Key}} \ \wedge \\ \nexists (u, \mathsf{m}, \mathsf{ADM}) \in \mathcal{Q}^{\mathsf{Sig}} : \mathsf{m}^\star \overset{\text{ADM}}{\preceq} \mathsf{m}))\end{array}\right] \leq \varepsilon(\kappa).$$

Designated-verifier unforgeability models the requirement that a designated-verifier signature can only be obtained in two ways: either by corretly redacting a signature (which can be done by everybody having access to the latter), or by having access to the secret key of the designated verifier. The former option would be chosen whenever a signature is to be legitimately forwarded to a receiver, while the latter enables the designated verifier to fake signatures.

Together with the previous definition, designated-verifier unforgeability guarantees that no adversary can come up with a designated-verifier signature for a foreign public key: by Definition 8 it is infeasible to forge a signature—and Definition 9 states that the only way of generating a designated-verifier signature for somebody else is to know a valid signature to start from.

Definition 9. *An* AD-RS *is designated-verifier unforgeable, if there exists a PPT opener* $\mathsf{O} = (O_1, O_2)$ *such that for every PPT adversary* $\mathcal{A}$ *there is a negligible function* $\varepsilon_1(\cdot)$ *such that*

$$\left|\begin{array}{l}\Pr\left[(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\kappa, n) : \mathcal{A}(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}) = 1\right] \ - \\ \Pr\left[(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}, \tau) \leftarrow O_1(1^\kappa, n) : \mathcal{A}(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}) = 1\right]\end{array}\right| \leq \varepsilon_1(\kappa),$$

and for every PPT adversary $\mathcal{A}$ *there is a negligible function* $\varepsilon_2(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}, \tau) \leftarrow O_1(1^\kappa, n), \\ \mathcal{O} \leftarrow \{\mathsf{Sig}(\cdot,\cdot,\cdot), \mathsf{Key}(\cdot), \\ \mathsf{DVGen}(\cdot), \mathsf{DVKey}(\cdot), \\ \mathsf{Sim}(\cdot,\cdot,\cdot,\cdot,\cdot)\}, \\ (\mathsf{m}^\star, \rho^\star, v^\star) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{gpk}, \mathsf{gok}), \\ u \leftarrow O_2(\tau, \mathtt{DVK}, \mathsf{m}^\star, \rho^\star, v^\star)\end{array} : \begin{array}{r}\mathsf{Verify}(\mathsf{gpk}, \mathsf{vpk}_{v^\star}, \mathsf{m}^\star, \rho^\star) = 1 \ \wedge \\ v^\star \notin \mathcal{Q}^{\mathsf{DVKey}} \ \wedge \\ \wedge \ (u = \bot \ \vee \ (u \notin \mathcal{Q}^{\mathsf{Key}} \ \wedge \\ \nexists (u, \mathsf{m}, \mathsf{ADM}) \in \mathcal{Q}^{\mathsf{Sig}} : \mathsf{m}^\star \overset{\text{ADM}}{\preceq} \mathsf{m})) \ \wedge \\ \nexists (v^\star, \mathsf{m}, \mathsf{ADM}, \cdot, \cdot) \in \mathcal{Q}^{\mathsf{Sim}} : \mathsf{m}^\star \overset{\text{ADM}}{\preceq} \mathsf{m})\end{array}\right] \leq \varepsilon_2(\kappa).$$

In our definition, we assume a simple key registration for designated verifiers to ensure that all designated-verifier key pairs have been honestly created and thus an adversary is not able to mount rogue key attacks. In practice, this requirement can often be alleviated by introducing an option to check the honest generation of the keys (cf. [RY07]), which we omit for simplicity.

Simulatability captures that designated verifiers can simulate signatures on arbitrary messages which are indistinguishable from honestly computed signatures.

Definition 10. *An* AD-RS *satisfies the simulatability property, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\varepsilon(\cdot)$ such that it holds that*

$$\Pr\left[\begin{array}{l} (\mathsf{gpk},\mathsf{gok},\mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\kappa, n),\ b \xleftarrow{R} \{0,1\}, \\ \mathcal{O} \leftarrow \{\mathsf{DVGen}(\cdot), \mathsf{DVKey}(\cdot)\}, \\ ((\mathsf{m}_0, \mathsf{ADM}_0, \mathsf{MOD}_0), (\mathsf{m}_1, \mathsf{ADM}_1), \\ i^\star, j^\star, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{gpk},\mathsf{gok},\mathsf{gsk}), \\ \sigma = (\underline{\sigma}, \overline{\sigma}) \leftarrow \mathsf{Sign}(\mathsf{gsk}_{i^\star}, \mathsf{m}_b, \mathsf{ADM}_b), \\ (\mathring{\mathsf{m}}_0, \rho) \leftarrow \mathsf{RoS}(b, j^\star, \mathsf{m}_0, \mathsf{ADM}_0, \mathsf{MOD}_0, \sigma), \\ b^\star \leftarrow \mathcal{A}^{\mathcal{O}}(\underline{\sigma}, \mathring{\mathsf{m}}_0, \rho, \mathsf{st}) \end{array} \;\middle|\; \begin{array}{r} b = b^\star \ \wedge \\ : \ \mathsf{ADM}_0 \preceq \mathsf{m}_0 \ \wedge \\ \mathsf{ADM}_1 \preceq \mathsf{m}_1 \end{array}\right] \leq 1/2 + \epsilon(\kappa).$$

As mentioned earlier, we assume that signatures consist of a private and a public component (the latter being denoted by $\underline{\sigma}$). To eliminate potential privacy issues associated with a public $\underline{\sigma}$, we also give $\underline{\sigma}$ as input to the simulator and the adversary, and require that the adversary cannot tell real and faked signatures apart *even when knowing* $\underline{\sigma}$. This way, our definitional framework guarantees that these parts do not contain any sensitive information.

In a realization of the system, the public parts of all signatures issued by the hospital would be made publicly available (without further meta-information).

Signer anonymity requires that only the opening authority can determine the identity of a signer.

Definition 11. *An* AD-RS *is signer anonymous, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\varepsilon(\cdot)$ such that*

$$\Pr\left[\begin{array}{l} (\mathsf{gpk},\mathsf{gok},\mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\kappa, n), \\ b \xleftarrow{R} \{0,1\}, \mathcal{O} \leftarrow \{\mathsf{Open}(\cdot,\cdot)\}, \\ (i_0^\star, i_1^\star, \mathsf{m}^\star, \mathsf{ADM}^\star, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{gpk},\mathsf{gsk}), \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{gsk}_{i_b^\star}, \mathsf{m}^\star, \mathsf{ADM}^\star), \\ b^\star \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma, \mathsf{st}) \end{array} \;\middle|\; \begin{array}{r} b = b^\star \ \wedge \\ : \nexists(\mathsf{m}, (\underline{\sigma}, \cdot)) \in \mathcal{Q}_2^{\mathsf{Open}} : \\ \mathsf{m} \overset{\mathsf{ADM}}{\preceq} \mathsf{m}^\star \end{array}\right] \leq 1/2 + \varepsilon(\kappa),$$

and $\mathcal{A}$ runs in two stages and $\mathcal{Q}_2^{\mathsf{Open}}$ records queries to oracle Open *in stage two.*

The definition guarantees that—no matter how many signatures already have been opened—the signers' identities for all other signatures remain secret. The formulation is, up to the last clause of the winning condition, similar to the anonymity definition of group signature schemes (cf. Definition 5). We, however, need to adapt the last clause because Definition 5 requires signatures to be

non-malleable. In contrast, our signatures are malleable by definition. However, we can still require parts of the signature, and in particular the public part, to be non-malleable. By doing so, we can achieve a strong notion that resembles anonymity in the sense of group signatures whenever honestly generated signatures have different public components with overwhelming probability. This is in particular the case for our instantiations provided in the next sections.

Privacy guarantees that a redacted designated-verifier signature does not leak anything about the blacked-out parts of the original message.

Definition 12. *An* AD-RS *is private, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}(\mathsf{gpk},\mathsf{gok},\mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\kappa, n),\ b \xleftarrow{R} \{0,1\},\\ \mathcal{O} \leftarrow \{\mathsf{Sig}(\cdot,\cdot,\cdot), \mathsf{Ch}(\cdot,\cdot,\cdot,\cdot,b)\}, \qquad : b = b^\star\\ b^\star \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{gpk},\mathsf{gok},\mathsf{gsk})\end{array}\right] \leq 1/2 + \varepsilon(\kappa).$$

We call an AD-RS *secure*, if it is correct, group unforgeable, designated-verifier unforgeable, simulatable, signer anonymous, and private.

Group Redactable Signatures. When omitting the DV-related notions and oracles, one directly obtains a definition of group redactable signatures, which may also be useful for applications that require revocable signer-anonymity.

4 A Generic Construction

Now we present a simple generic construction which can be built by combining any GS, any RS, and any Π that admits proofs of knowledge in a black-box way. In Scheme 1 we present our construction which follows the intuition given in the introduction. We use Π to prove knowledge of a witness for the following **NP** relation R required by the verification of designated-verifier signatures.

$$\begin{aligned}((\mathsf{m},\mathsf{pk},\mathsf{vpk}_j),\ (\sigma_\mathsf{R},\sigma_\mathsf{V})) \in R \iff\\ \mathsf{RS.Verify}(\mathsf{pk},\mathsf{m},\sigma_\mathsf{R}) = 1 \ \vee\ \Sigma.\mathsf{Verify}(\ vpk_j,\mathsf{m},\sigma_\mathsf{V}) = 1.\end{aligned}$$

The rationale behind choosing R in this way is that this yields the most general result. That is, no further assumptions on RS or Σ are required.

Theorem 1 (proven in Appendix B). *If* GS, RS, *and* Σ *are secure and* Π *is witness indistinguishable and admits proofs of knowledge, then Scheme 1 is secure.*

For an instantiation of our construction we can use standard GS and standard RS, where multiple practically efficient instantiations exist. Thus, the time required for signature creation/verification is mainly determined by the cost of the proof of knowledge of the RS signature σ_R. We, however, want to emphasize that—depending on the concrete RS—this proof can usually be instantiated by means of relatively cheap Σ-protocols. Ultimately, as we will show below, we can replace this proof with a much cheaper proof by exploiting properties of the used RS.

$\mathsf{Setup}(1^\kappa, n)$: Run $(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}) \leftarrow \mathsf{GS.KeyGen}(1^\kappa, n)$, $\mathsf{crs} \leftarrow \Pi.\mathsf{Setup}(1^\kappa)$, set $\mathsf{gpk}' \leftarrow (\mathsf{gpk}, \mathsf{crs})$ and return $(\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk})$.

$\mathsf{DVGen}(1^\kappa)$: Run $(\mathsf{vsk}_j, \mathsf{vpk}_j) \leftarrow \Sigma.\mathsf{KeyGen}(1^\kappa)$ and return $(\mathsf{vsk}_j, \mathsf{vpk}_j)$.

$\mathsf{Sign}(\mathsf{gsk}_i, \mathsf{m}, \textsc{adm})$: Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{RS.KeyGen}(1^\kappa)$ and return $\sigma = (\underline{\sigma}, \overline{\sigma}) \leftarrow ((\mathsf{pk}, \sigma_\mathsf{G}), (\sigma_\mathsf{R}, \textsc{red}))$, with

$$\sigma_\mathsf{G} \leftarrow \mathsf{GS.Sign}(\mathsf{gsk}_i, \mathsf{pk}), \text{ and } ((\mathsf{m}, \sigma_\mathsf{R}), \textsc{red}) \leftarrow \mathsf{RS.Sign}(\mathsf{sk}, \mathsf{m}, \textsc{adm}).$$

$\mathsf{GVerify}(\mathsf{gpk}, \mathsf{m}, \sigma)$: Parse σ as $((\mathsf{pk}, \sigma_\mathsf{G}), (\sigma_\mathsf{R}, \cdot))$ and return 1 if the following holds and 0 otherwise:

$$\mathsf{GS.Verify}(\mathsf{gpk}, \mathsf{pk}, \sigma_\mathsf{G}) = 1 \quad \wedge \quad \mathsf{RS.Verify}(\mathsf{pk}, \mathsf{m}, \sigma_\mathsf{R}) = 1.$$

$\mathsf{Open}(\mathsf{gok}, \mathsf{m}, \sigma)$: Parse σ as $((\mathsf{pk}, \sigma_\mathsf{G}), \overline{\sigma})$ and return $\mathsf{GS.Open}(\mathsf{gok}, \mathsf{pk}, \sigma_\mathsf{G})$.

$\mathsf{Redact}(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \sigma, \textsc{mod})$: Parse σ as $((\mathsf{pk}, \sigma_\mathsf{G}), (\sigma_\mathsf{R}, \textsc{red}))$ and return $(\mathring{\mathsf{m}}, \rho)$, where

$$((\mathring{\mathsf{m}}, \mathring{\sigma}_\mathsf{R}), \cdot) \leftarrow \mathsf{RS.Redact}(\mathsf{pk}, \mathsf{m}, \sigma_\mathsf{R}, \textsc{mod}, \textsc{red}),$$
$$\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathring{\mathsf{m}}, \mathsf{pk}, \mathsf{vpk}_j), (\mathring{\sigma}_\mathsf{R}, \perp)), \text{ and}$$
$$\rho \leftarrow ((\mathsf{pk}, \sigma_\mathsf{G}), \pi).$$

$\mathsf{Verify}(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \rho)$: Parse ρ as $((\mathsf{pk}, \sigma_\mathsf{G}), \pi)$ and return 1 if the following holds, and 0 otherwise:

$$\mathsf{GS.Verify}(\mathsf{gpk}, \mathsf{pk}, \sigma_\mathsf{G}) = 1 \quad \wedge \quad \Pi.\mathsf{Verify}(\mathsf{crs}, (\mathsf{m}, \mathsf{pk}, \mathsf{vpk}_j), \pi) = 1.$$

$\mathsf{Sim}(\mathsf{gpk}, \mathsf{vsk}_j, \mathsf{m}, \textsc{adm}, \textsc{mod}, \underline{\sigma})$: If $\textsc{mod} \preceq \textsc{adm} \wedge \textsc{adm} \preceq \mathsf{m}$, parse $\underline{\sigma}$ as $(\mathsf{pk}, \sigma_\mathsf{G})$, run $\mathring{\mathsf{m}} \xleftarrow{\textsc{mod}} \mathsf{m}$, and return $(\mathring{\mathsf{m}}, \rho)$, where

$$\sigma_\mathsf{V} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{vsk}_j, \mathring{\mathsf{m}}),$$
$$\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathring{\mathsf{m}}, \mathsf{pk}, \mathsf{vpk}_j), (\perp, \sigma_\mathsf{V})), \text{ and}$$
$$\rho \leftarrow (\underline{\sigma}, \pi).$$

Otherwise, return $\perp$.

Scheme 1: Black-Box AD-RS

5 Boosting Efficiency via Key-Homomorphisms

In [DPSS15] it is shown that RS can be generically constructed from any EUF-CMA secure signature scheme and indistinguishable accumulators [DHS15]. In our setting it is most reasonable to consider messages as an (ordered) sequence of message blocks. A straight forward solution would thus be to build upon [DPSS15, Scheme 2], which is tailored to signing ordered sequences of messages $\mathsf{m} = (m_1, \ldots, m_n)$. Unfortunately, this construction aims to conceal the number of message blocks in the original message, and the positions of the redactions. This can be dangerous in our setting, since it might allow to completely change the document semantics. Besides that, it inherently requires a more complex construction.

To this end, we pursue a different direction and require another message representation: we make the position i of the message blocks m_i in the message explicit and represent messages as sets $\mathsf{m} = \{1||m_1, \ldots, n||m_n\}$. Besides solving the aforementioned issues, it also allows us to build upon the (simpler) RS paradigm for sets [DPSS15, Scheme 1]. This paradigm subsumes the essence of many existing RSs and works as follows. Secret keys, public keys, and signatures are split into two parts each. One corresponds to the signature scheme Σ, and one corresponds to the accumulator Λ. Then, Λ is used to encode the message, whereas Σ is used to sign the encoded message. Consequently, we can look at RS key pairs and signatures as being of the form $(\mathsf{sk}, \mathsf{pk}) = ((\mathsf{sk}_\Sigma, \mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), (\mathsf{pk}_\Sigma, \mathsf{pk}_\Lambda))$ and $\sigma_\mathsf{R} = (\sigma_\Sigma, \sigma_\Lambda)$ where the indexes denote their respective types. We emphasize that for accumulators it holds by definition that sk_Λ is an optional trapdoor which may enable more efficient computations, but all algorithms also run without sk_Λ and the output distribution of the algorithms does not depend on whether the algorithms are executed with or without sk_Λ [DHS15, DPSS15]. We require this property to be able to create designated verifier signatures (cf. Sim) and use $(\mathsf{sk}_\Sigma, \perp, \mathsf{pk}_\Lambda)$ to denote an RS secret key without sk_Λ.

RS following this paradigm only require Σ (besides correctness) to be EUF-CMA secure. We observe that additional constraints on Σ—and in particular the key-homomorphism as we define it below—does not influence RS security, while it enables us to design the relation R such that it admits very efficient proofs.

Key-Homomorphic Signatures. Informally, we require signature schemes where, for a given public key and a valid signature under that key, one can adapt the public key and the signature so that the resulting signature is valid with respect to the initial message under the new public key. Moreover, adapted signatures need to be identically distributed as fresh signatures under the secret key corresponding to the adapted public key.

Key-malleability in the sense of adapting given signatures to other signatures under related keys has so far mainly been studied in context of related-key attacks (RKAs) [BCM11], where one aims to rule out such constructions. Signatures with re-randomizable keys which allow to consistently update secret and public keys, but without considering adaption of existing signatures, have recently been introduced and studied in [FKM+16]. As we are not aware of any constructive use of and definitions for the functionality we require, we define key-homomorphic signatures inspired by key-homomorphic symmetric encryption (cf. [AHI11]).

Let $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be an EUF-CMA secure signature scheme where the secret and public keys live in groups $(\mathbb{H}, +)$ and $(\mathbb{G}, \cdot)$, respectively. Inspired by the definition for encryption schemes in [TW14], we define the following.

Definition 13 (Secret-Key to Public-Key Homomorphism). *A signature scheme Σ provides a secret-key to public-key homomorphism, if there exists an efficiently computable map $\mu : \mathbb{H} \to \mathbb{G}$ such that for all $\mathsf{sk}, \mathsf{sk}' \in \mathbb{H}$ it holds that $\mu(\mathsf{sk} + \mathsf{sk}') = \mu(\mathsf{sk}) \cdot \mu(\mathsf{sk}')$, and for all $(\mathsf{sk}, \mathsf{pk})$ output by KeyGen, it holds that $\mathsf{pk} = \mu(\mathsf{sk})$.*

Now, we define key-homomorphic signatures, where we focus on the class of functions Φ^+ representing linear shifts. We stress that Φ^+ is a finite set of functions, all with the same domain and range, and, in our case depends on the public key of the signature scheme (which is not made explicit). Moreover, Φ^+ admits an efficient membership test and its functions are efficiently computable.

Definition 14 (Φ^+-Key-Homomorphic Signatures). *A signature scheme is called Φ^+-key-homomorphic, if it provides a secret-key to public-key homomorphism and an additional PPT algorithm* Adapt*, defined as:*

$\mathsf{Adapt}(\mathsf{pk}, \mathsf{m}, \sigma, \Delta)$: *Takes a public key* pk*, a message* m*, a signature* σ*, and a function* $\Delta \in \Phi^+$ *as input, and outputs a public key* pk' *and a signature* σ'*,*

where for all $\Delta \in \Phi^+$*, all* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*, all messages* m*, all* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$*, all* $(\mathsf{pk}', \sigma') \leftarrow \mathsf{Adapt}(\mathsf{pk}, \mathsf{m}, \sigma, \Delta)$ *it holds that* $\mathsf{Verify}(\mathsf{pk}', \mathsf{m}, \sigma') = 1$ *and* $\mathsf{pk}' = \Delta(\mathsf{pk})$*.*

For simplicity we sometimes identify a function $\Delta \in \Phi^+$ with its "shift amount" $\Delta \in \mathbb{H}$. To model that freshly generated signatures look identical as adapted signatures on the same message, we introduce the following additional property.

Definition 15 (Adaptability of Signatures). *A Φ^+-key-homomorphic signature scheme provides adaptability of signatures, if for every $\kappa \in \mathbb{N}$, and every message* m*, it holds that* $\mathsf{Adapt}(\mathsf{pk}, \mathsf{m}, \mathsf{Sign}(\mathsf{sk}, \mathsf{m}), \Delta)$ *and* $(\mathsf{pk} \cdot \mu(\Delta), \mathsf{Sign}(\mathsf{sk} + \Delta, \mathsf{m}))$ *as well as* $(\mathsf{sk}, \mathsf{pk})$ *and* $(\mathsf{sk}', \mu(\mathsf{sk}'))$ *are identically distributed, where* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*,* $\mathsf{sk}' \xleftarrow{R} \mathbb{H}$*, and* $\Delta \xleftarrow{R} \Phi^+$*.*

For an in-depth treatment and examples of key-homomorphic signatures, we refer the reader to a more recent work [DS16b]. The important bottom-line here is that there are various efficient schemes that satisfy Definition 15. For instance, Schnorr signatures [Sch91], BLS signatures [BLS04], the recent scheme by Pointcheval and Sanders [PS16] or Waters signatures [Wat05].

Φ^+-Key-Homomorphic Redactable Signature Schemes. When instantiating the RS construction paradigm from [DPSS15] (as outlined above) with a Φ^+-key-homomorphic signature scheme, the key homomorphism of the signature scheme straight-forwardly carries over to the RS and we can define Adapt as follows.

$\mathsf{Adapt}(\mathsf{pk}, \mathsf{m}, \sigma, \Delta)$: Parse pk as $(\mathsf{pk}_\Sigma, \mathsf{pk}_\Lambda)$ and σ as $(\sigma_\Sigma, \sigma_\Lambda)$, run $(\mathsf{pk}'_\Sigma, \sigma'_\Sigma) \leftarrow \mathsf{Adapt}(\mathsf{pk}_\Sigma, \Lambda(\mathsf{m}), \sigma_\Sigma, \Delta)$ and return $(\mathsf{pk}', \sigma') \leftarrow ((\mathsf{pk}'_\Sigma, \mathsf{pk}_\Lambda), (\sigma'_\Sigma, \sigma_\Lambda))$.

This allows us to concisely present our construction in Scheme 2. The **NP** relation, which needs to be satisfied by valid designated-verifier signatures is as follows.

$$((\mathsf{pk}, \mathsf{vpk}_j), (\mathsf{sk}, \mathsf{vsk}_j)) \in R \iff \mathsf{pk} = \mu(\mathsf{sk}) \ \vee \ \Sigma.\mathsf{VKey}(\mathsf{vsk}_j, \mathsf{vpk}_j) = 1.$$

In the discrete logarithm setting such a proof requires an OR-Schnorr proof of two discrete logs, i.e., only requires *two group exponentiations*.

$\mathsf{Redact}(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \sigma, \text{MOD})$: Parse σ as $((\mathsf{pk}, \sigma_\mathsf{G}), (\sigma_\mathsf{R}, \text{RED}))$ and return $(\mathring{\mathsf{m}}, \rho)$, where

$$\mathsf{sk}' \xleftarrow{R} \mathbb{H},\ \mathsf{pk}' \leftarrow \mu(\mathsf{sk}'),\ (\mathsf{pk}_\mathsf{R}, \sigma'_\mathsf{R}) \leftarrow \mathsf{Adapt}(\mathsf{pk}, \mathsf{m}, \sigma_\mathsf{R}, \mathsf{sk}'),$$
$$((\mathring{\mathsf{m}}, \mathring{\sigma}'_\mathsf{R}), \cdot) \leftarrow \mathsf{RS.Redact}(\mathsf{pk}_\mathsf{R}, \mathsf{m}, \sigma'_\mathsf{R}, \text{MOD}, \text{RED}),$$
$$\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathsf{pk}', \mathsf{vpk}_j), (\mathsf{sk}', \bot)),\ \text{and}\ \rho \leftarrow ((\mathsf{pk}, \sigma_\mathsf{G}), \mathsf{pk}', \mathring{\sigma}'_\mathsf{R}, \pi).$$

$\mathsf{Verify}(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \rho)$: Parse ρ as $((\mathsf{pk}, \sigma_\mathsf{G}), \mathsf{pk}', \mathring{\sigma}'_\mathsf{R}, \pi)$, let $\mathsf{pk} = (\mathsf{pk}_\Sigma, \mathsf{pk}_\Lambda)$, compute $\mathsf{pk}_\mathsf{R} \leftarrow (\mathsf{pk}_\Sigma \cdot \mathsf{pk}', \mathsf{pk}_\Lambda)$ and return 1 if the following holds, and 0 otherwise:

$$\mathsf{GS.Verify}(\mathsf{gpk}, \mathsf{pk}, \sigma_\mathsf{G}) = 1 \quad \wedge \quad \Pi.\mathsf{Verify}(\mathsf{crs}, (\mathsf{pk}', \mathsf{vpk}_j), \pi) = 1$$
$$\wedge \quad \mathsf{RS.Verify}(\mathsf{pk}_\mathsf{R}, \mathsf{m}, \mathring{\sigma}'_\mathsf{R}) = 1.$$

$\mathsf{Sim}(\mathsf{gpk}, \mathsf{vsk}_j, \mathsf{m}, \text{ADM}, \text{MOD}, \underline{\sigma})$: If $\text{MOD} \preceq \text{ADM} \wedge \text{ADM} \preceq \mathsf{m}$, parse $\underline{\sigma}$ as $((\mathsf{pk}_\Sigma, \mathsf{pk}_\Lambda), \sigma_\mathsf{G})$ and return $(\check{\mathsf{m}}, \rho)$, where

$$\mathsf{sk}^\Sigma_\mathsf{R} \xleftarrow{R} \mathbb{H},\ \mathsf{pk}^\Sigma_\mathsf{R} \leftarrow \mu(\mathsf{sk}^\Sigma_\mathsf{R}),\ \mathsf{pk}' \leftarrow \mathsf{pk}_\Sigma^{-1} \cdot \mathsf{pk}^\Sigma_\mathsf{R},$$
$$((\mathsf{m}, \sigma'_\mathsf{R}), \text{RED}) \leftarrow \mathsf{RS.Sign}((\mathsf{sk}^\Sigma_\mathsf{R}, \bot, \mathsf{pk}_\Lambda), \mathsf{m}, \text{ADM}),$$
$$((\mathring{\mathsf{m}}, \mathring{\sigma}'_\mathsf{R}), \cdot) \leftarrow \mathsf{RS.Redact}((\mathsf{pk}^\Sigma_\mathsf{R}, \mathsf{pk}_\Lambda), \mathsf{m}, \sigma'_\mathsf{R}, \text{MOD}, \text{RED}),$$
$$\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathsf{pk}', \mathsf{vpk}_j), (\bot, \mathsf{vsk}_j)),\ \text{and}\ \rho \leftarrow (\underline{\sigma}, \mathsf{pk}', \mathring{\sigma}'_\mathsf{R}, \pi).$$

Scheme 2: Semi-Black-Box AD-RS where Setup, DVGen, Sign, GVerify, and Open are as in Scheme 1.

Theorem 2 (proven in Appendix C). *If* GS *is secure,* RS *is an adaptable* RS *following [DPSS15, Scheme 1],* Σ *is secure, and* Π *is weakly simulation sound extractable, then Scheme 2 is also secure.*

5.1 Performance Overview

In this section we evaluate the practical efficiency of Scheme 2. We first assess the practicality of the underlying components and then analyze the overhead imposed by the provably provided strong security guarantees.

Group Signatures. It is well known that there exist multiple practically efficient group signature schemes for non-constrained devices such as standard PCs or even more powerful machines in the cloud. Yet, to adequately protect the doctor's group signing key—which is the only key that persists over multiple signing operations—it might make sense to compute the doctor's group signature σ_G on the one-time RS public key pk upon Sign on some dedicated signature token such as a smart card or smart phone. Using the estimations in [DS16a], such a signature can be computed in $\approx$ 1s on an ARM Cortex-M0+, a processor that is small enough to be employed in smart cards. While this is already acceptable, the performance on smart phones will even be significantly better. For instance, [CDDT12] report execution times of approximately 150ms for the computation of a group signature with the well-known BBS [BBS04] scheme on a by now rather outdated smart phone.

Key-Homomorphic Redactable Signatures. We first note that the RS keys are freshly generated and the secret keys can be deleted after each signing operation. The respective operations can therefore be directly executed on the doctor's PC, potentially even in parallel to the computation of the group signature. Since we are not aware of any performance evaluation of RS on standard PCs, we implemented one possible instantiation of [DPSS15, Scheme 1]. In particular, we based our RS implementation on Schnorr signatures and the indistinguishable t-SDH accumulator from [DHS15] without further optimizations regarding efficiency. In Table 1, we present our performance results on an Intel Core i7-4790 @ 3.60GHz with 8GB of RAM, running Java 1.8.0_91 on top of Ubuntu 16.04. Each value represents the mean of 100 consecutive executions. These results confirm that the required RS paradigm is perfectly suited for our application.

Sign	Verify	Redact	Verify (after Redact)
73.1ms	886.3ms	0.1ms	450.3ms

Table 1. RS timings in milliseconds, with a number of $n = 100$ message blocks, 50% admissibly redactable blocks and 25% of the blocks being redacted upon Redact.

Additional Computations. Using Schnorr signatures, one only needs two group exponentiations for the proof of knowledge[5]; the adaption of the signature only requires a $\mathbb{Z}_p$ operation, which, compared to the group exponentiations, can be neglected. All in all, the additional computations can thus be ignored compared to those of GS and RS[6], even on very constrained devices such as [UW14].

Signature Size. Regarding signature size, the dominant part is the size of the RS public key and signature, respectively, which is in turn determined by the choice of the accumulator. In particular, when instantiating the RS with an accumulator having constant key size and supporting batch verification, one can even obtain constant size signatures. We refer the reader to [DPSS15] for a discussion on RS signature sizes and [DHS15] for an overview of suitable accumulators.

6 Conclusion

We introduce the notion of signer-anonymous designated-verifier redactable signatures, extending redactable signatures in their vanilla form in several important directions. These additional features are motivated by a real world use-case

[5] The results of [FKMV12] confirm that one can use Fiat-Shamir transformed Σ-protocols in the discrete log setting as simulation sound extractable (and therefore weakly simulation sound extractable) proof system when including the statement x upon computing the hash for the challenge.

[6] This is underpinned by the results in Table 1, where $\mathcal{O}(n)$ exponentiations happen.

in the health care field, demonstrating its practical relevance. Besides rigorously modelling this primitive, we provide two instantiations. While both are interesting from a theoretical point of view, the latter is also interesting in practice. In particular, due to using key-homomorphic signatures as we introduce them in this paper, we obtain a simple and practically efficient solution.

References

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In *ICS*, 2011.

[BBD+10] Christina Brzuska, Heike Busch, Özgür Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In *ACNS*, 2010.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, 2004.

[BCM11] Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In *ASIACRYPT*, 2011.

[BFLS10] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In *PKC*, 2010.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17(4), 2004.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT*, 2003.

[BPS13] Christina Brzuska, Henrich Christopher Pöhls, and Kai Samelin. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In *EuroPKI 2013*, 2013.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, 2005.

[CA89] David Chaum and Hans Van Antwerpen. Undeniable signatures. In *CRYPTO*, 1989.

[CDDT12] Sébastien Canard, Nicolas Desmoulins, Julien Devigne, and Jacques Traoré. On the implementation of a pairing-based cryptographic protocol in a constrained device. In *Pairing*, 2012.

[CDHK15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In *ASIACRYPT*, 2015.

[Cha94] David Chaum. Designated confirmer signatures. In *EUROCRYPT*, 1994.

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, 2015.

[DPSS15] David Derler, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. A general framework for redactable signatures and new constructions. In *ICISC*, 2015.

[DS16a] David Derler and Daniel Slamanig. Fully-anonymous short dynamic group signatures without encryption, 2016.

[DS16b] David Derler and Daniel Slamanig. Key-homomorphic signatures and applications to multiparty signatures. *IACR Cryptology ePrint Archive*, 2016:792, 2016.

[FKM+16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *PKC*, 2016.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *INDOCRYPT*, 2012.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO '86*, volume 263, pages 186–194, 1986.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, 2006.

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA*, 2002.

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In *EUROCRYPT*, 1996.

[LWB05] Helger Lipmaa, Guilin Wang, and Feng Bao. Designated verifier signature schemes: Attacks, new security notions and a new construction. In *ICALP*, 2005.

[LZCS16] Russell W. F. Lai, Tao Zhang, Sherman S. M. Chow, and Dominique Schröder. Efficient sanitizable signatures without random oracles. In *ESORICS*, 2016.

[MPV09] Jean Monnerat, Sylvain Pasini, and Serge Vaudenay. Efficient deniable authentication for signatures. In *ACNS*, 2009.

[PS15] Henrich C. Pöhls and Kai Samelin. Accountable redactable signatures. In *ARES*, 2015.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *CT-RSA*, 2016.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT*, 2007.

[SBWP03] Ron Steinfeld, Laurence Bull, Huaxiong Wang, and Josef Pieprzyk. Universal designated-verifier signatures. In *ASIACRYPT*, 2003.

[SBZ01] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In *ICISC*, 2001.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[SS08] Siamak Fayyaz Shahandashti and Reihaneh Safavi-Naini. Construction of universal designated-verifier signatures and identity-based signatures from standard signatures. In *PKC*, 2008.

[TW14] Stefano Tessaro and David A. Wilson. Bounded-collusion identity-based encryption from semantically-secure public-key encryption: Generic constructions with short ciphertexts. In *PKC*, 2014.

[UW14] Thomas Unterluggauer and Erich Wenger. Efficient pairings and ECC for embedded systems. In *CHES*, 2014.

[Ver06] Damien Vergnaud. New extensions of pairing-based signatures into universal designated verifier signatures. In *ICALP*, 2006.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, 2005.

A Definitions and Security Notions

Digital Signatures. Subsequently, we recall a definition of signatures.

Definition 16. *A signature scheme* Σ *is a triple* (KeyGen, Sign, Verify) *of PPT algorithms, which are defined as follows:*

$\mathsf{KeyGen}(1^\kappa)$: *This algorithm takes a security parameter* κ *as input and outputs a secret (signing) key* sk *and a public (verification) key* pk *with associated message space* $\mathcal{M}$ *(we may omit to mention the message space* $\mathcal{M}$*).*

$\mathsf{Sign}(\mathsf{sk}, m)$: *This algorithm takes a secret key* sk *and a message* $m \in \mathcal{M}$ *as input and outputs a signature* σ*.*

$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: *This algorithm takes a public key* pk*, a message* $m \in \mathcal{M}$ *and a signature* σ *as input and outputs a bit* $b \in \{0, 1\}$*.*

In addition, we require an algorithm $\mathsf{VKey}(\cdot, \cdot)$, which checks whether a key pair is a valid output of KeyGen, i.e., for any $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ we have $\mathsf{VKey}(\mathsf{sk}, \mathsf{pk}) = 1$. Besides correctness, Σ needs to be existentially unforgeable under adaptively chosen message attacks (EUF-CMA). Subsequently, we formally recall the definition of EUF-CMA security.

Definition 17 (EUF-CMA)**.** *A signature scheme* Σ *is* EUF-CMA *secure, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\left[\begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa), \\ (m^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{pk})\end{array} : \begin{array}{r}\mathsf{Verify}(\mathsf{pk}, m^\star, \sigma^\star) = 1 \ \wedge \\ m^\star \notin \mathcal{Q}^{\mathsf{Sign}}\end{array}\right] \leq \varepsilon(\kappa),$$

where the environment keeps track of the queries to the signing oracle via $\mathcal{Q}^{\mathsf{Sign}}$*.*

We call a signature scheme *secure*, if it is correct and provides EUF-CMA security.

Non-Interactive Proof Systems. Now, we recall a standard definition of non-interactive proof systems (Π). Our definitions are inspired by [Gro06]. Therefore, let L_R be an **NP**-language with witness relation R defined as $L_R = \{x \mid \exists\ w : R(x, w) = 1\}$.

Definition 18. *A non-interactive proof system* Π *is a tuple of algorithms* (Setup, Proof, Verify)*, which are defined as follows:*

$\mathsf{Setup}(1^\kappa)$: This PPT algorithm takes a security parameter κ as input, and outputs a common reference string crs.

$\mathsf{Proof}(\mathsf{crs}, x, w)$: This algorithm takes a common reference string crs, a statement x, and a witness w as input, and outputs a proof π.

$\mathsf{Verify}(\mathsf{crs}, x, \pi)$: This PPT algorithm takes a common reference string crs, a statement x, and a proof π as input, and outputs a bit $b \in \{0, 1\}$.

If, in addition, if algorithm Proof runs in polynomial time, then we talk about a non-interactive witness-indistinguishable argument system. We require Π to be complete, sound, and adaptively witness-indistinguishable. Subsequently, we recall formal definition of those properties.

Definition 19 (Completeness). *A non-interactive proof system* Π *is* complete*, if for every adversary* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{l}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa),\ (x^\star, w^\star) \leftarrow \mathcal{A}(\mathsf{crs}),\\ \pi \leftarrow \mathsf{Proof}(\mathsf{crs}, x^\star, w^\star)\end{array} : \begin{array}{r}\mathsf{Verify}(\mathsf{crs}, x^\star, \pi) = 1\\ \wedge\ (x^\star, w^\star) \in R\end{array}\right] = 1.$$

Definition 20 (Soundness). *A non-interactive proof system* Π *is* sound*, if for every PPT adversary* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa),\ (x^\star, \pi^\star) \leftarrow \mathcal{A}(\mathsf{crs}) \ :\ \begin{array}{r}\mathsf{Verify}(\mathsf{crs}, x^\star, \pi^\star) = 1\\ \wedge\ x^\star \notin L_R\end{array}\right] \leq \varepsilon(\kappa).$$

If $\varepsilon = 0$, we have perfect soundness.

Definition 21 (Adaptive Witness-Indistinguishability). *A non-interactive proof system* Π *is* adaptively witness-indistinguishable*, if for every PPT adversary* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa),\ b \xleftarrow{R} \{0,1\},\ b^\star \leftarrow \mathcal{A}^{\mathcal{P}(\mathsf{crs},\cdot,\cdot,\cdot,b)}(\mathsf{crs}) : b = b^\star\right] \leq \varepsilon(\kappa),$$

where $\mathcal{P}(\mathsf{crs}, x, w_0, w_1, b) := \mathsf{Proof}(\mathsf{crs}, x, w_b)$*, and* $\mathcal{P}$ *returns* $\perp$ *if* $(x, w_0) \notin R \ \vee\ (x, w_1) \notin R$*.*

If $\varepsilon = 0$, we have perfect adaptive witness-indistinguishability.

Definition 22 (Adaptive Zero-Knowledge). *A non-interactive proof system* Π *is* adaptively zero-knowledge*, if there exists a PPT simulator* $S = (\mathsf{S}_1, \mathsf{S}_2)$ *such that for every adversary* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\left|\begin{array}{l}\Pr\left[\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa) \ :\ \mathcal{A}^{\mathcal{P}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1\right] \ -\\ \Pr\left[(\mathsf{crs}, \tau) \leftarrow \mathsf{S}_1(1^\kappa) \ :\ \mathcal{A}^{\mathcal{S}(\mathsf{crs},\tau,\cdot,\cdot)}(\mathsf{crs}) = 1\right]\end{array}\right| \leq \varepsilon(\kappa),$$

where, τ *denotes a simulation trapdoor. Thereby,* $\mathcal{P}$ *and* $\mathcal{S}$ *return* $\perp$ *if* $(x, w) \notin R$ *or* $\pi \leftarrow \mathsf{Proof}(\mathsf{crs}, x, w)$ *and* $\pi \leftarrow \mathsf{S}_2(\mathsf{crs}, \tau, x)$*, respectively, otherwise.*

If $\varepsilon = 0$, we have perfect adaptive zero-knowledge. It is easy to show that adaptive zero-knowledge implies adaptive witness indistinguishability.

Definition 23 (Proof of Knowledge). *A non-interactive proof system* Π *admits proofs of knowledge, if there exists a PPT extractor* $\mathsf{E} = (\mathsf{E}_1, \mathsf{E}_2)$ *such that for every PPT adversary* $\mathcal{A}$ *there is a negligible function* $\varepsilon_1(\cdot)$ *such that*

$$\left|\begin{array}{l}\Pr\left[\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa) \ :\ \mathcal{A}(\mathsf{crs}) = 1\right] \ -\\ \Pr\left[(\mathsf{crs}, \xi) \leftarrow \mathsf{E}_1(1^\kappa) \ :\ \mathcal{A}(\mathsf{crs}) = 1\right]\end{array}\right| \leq \varepsilon_1(\kappa),$$

and for every PPT adversary $\mathcal{A}$ *there is a negligible function* $\varepsilon_2(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}(\mathsf{crs}, \tau) \leftarrow \mathsf{E}_1(1^\kappa),\ (x^\star, \pi^\star) \leftarrow \mathcal{A}(\mathsf{crs}),\\ w \leftarrow \mathsf{E}_2(\mathsf{crs}, \xi, x^\star, \pi^\star)\end{array} : \begin{array}{r}\mathsf{Verify}(\mathsf{crs}, x^\star, \pi^\star) = 1\ \wedge\\ (x^\star, w) \notin R\end{array}\right] \leq \varepsilon_2(\kappa).$$

Definition 24 (Weak Simulation Sound Extractability). *An adaptively zero-knowledge non-interactive proof system* Π *is weakly simulation sound extractable, if there exists a PPT extractor* $\mathsf{E} = (\mathsf{S}, \mathsf{E})$ *such that for every adversary* $\mathcal{A}$ *it holds that*

$$\left| \begin{array}{l} \Pr\left[(\mathsf{crs}, \tau) \leftarrow \mathsf{S}_1(1^\kappa) \ : \ \mathcal{A}(\mathsf{crs}, \tau) = 1\right] \ - \\ \Pr\left[(\mathsf{crs}, \tau, \xi) \leftarrow \mathsf{S}(1^\kappa) \ : \ \mathcal{A}(\mathsf{crs}, \tau) = 1\right] \end{array} \right| = 0,$$

and for every PPT adversary $\mathcal{A}$ *there is a negligible function* $\varepsilon_2(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l} (\mathsf{crs}, \tau, \xi) \leftarrow \mathsf{S}(1^\kappa), \\ (x^\star, \pi^\star) \leftarrow \mathcal{A}^{\mathcal{S}(\mathsf{crs}, \tau, \cdot)}(\mathsf{crs}), \\ w \leftarrow \mathsf{E}(\mathsf{crs}, \tau, \xi, x^\star, \pi^\star) \end{array} \ : \ \begin{array}{r} \mathsf{Verify}(\mathsf{crs}, x^\star, \pi^\star) = 1 \ \wedge \\ (x^\star, \cdot) \notin \mathcal{Q}_\mathsf{S} \ \wedge \ (x^\star, w) \notin R \end{array}\right] \leq \varepsilon_2(\kappa),$$

where $\mathcal{S}(\mathsf{crs}, \tau, x) \coloneqq \mathsf{S}_2(\mathsf{crs}, \tau, x)$ *and* $\mathcal{Q}_\mathsf{S}$ *keeps track of the queries to and answers of* $\mathcal{S}$*.*

B Proof of Theorem 1

We show that Theorem 1 holds by proving Lemma 1-6.

Lemma 1. *If* GS *is correct,* RS *is correct,* Σ *is correct, and* Π *is complete, then Scheme 1 is also correct.*

Lemma 1 straight-forwardly follows from inspection; the proof is omitted.

Lemma 2. *If* GS *is traceable and* RS *is unforgeable, then Scheme 1 is group unforgeable.*

Proof. We construct efficient reductions $\mathcal{R}^\mathsf{t}$ and $\mathcal{R}^\mathsf{u}$ turning an efficient group unforgeability adversary $\mathcal{A}^{\mathsf{gu}}$, into an efficient adversary (1) $\mathcal{A}^\mathsf{t}$ against traceability of GS, or (2) $\mathcal{A}^\mathsf{u}$ against unforgeability of the RS.

(1) $\mathcal{R}^\mathsf{t}$ obtains $(\mathsf{gpk}, \mathsf{gok})$ from a GS traceability challenger $\mathcal{C}^\mathsf{t}_\kappa$, completes the setup as in the real game, and starts $\mathcal{A}^{\mathsf{gu}}$ on $(\mathsf{gpk}', \mathsf{gok})$. Sig queries are simulated by obtaining the group signature σ_G using the Sig oracle provided by $\mathcal{C}^\mathsf{t}_\kappa$ and running the remaining Sign algorithm as in the original protocol. Key queries are simply forwarded to $\mathcal{C}^\mathsf{t}_\kappa$. Eventually, $\mathcal{A}^{\mathsf{gu}}$ outputs a forgery $(\mathsf{m}^\star, \sigma^\star)$ which is opened to signer index u (recall that $\sigma^\star = ((\mathsf{pk}, \sigma_\mathsf{G}), (\sigma_\mathsf{R}, \mathsf{RED}))$. If u exists ($u \neq \perp$), and $\mathcal{A}^{\mathsf{gu}}$ either requested gsk_u or a group signature on pk for u (i.e., $u \in \mathcal{Q}^{\mathsf{Key}} \ \vee \ (u, \mathsf{pk}) \in \mathcal{Q}^{\mathsf{Sign}}$), we abort as we are in the other case. Otherwise, we output $(\mathsf{pk}, \sigma_\mathsf{G})$ as a valid forgery for traceability of GS.

(2) $\mathcal{R}^{\mathsf{u}}$ runs the setup as in the real game and starts $\mathcal{A}^{\mathsf{gu}}$ on $(\mathsf{gpk}', \mathsf{gok})$. On each Sig query, $\mathcal{R}^{\mathsf{u}}$ engages with an RS unforgeability challenger $\mathcal{C}^{\mathsf{u}}_{\kappa}$, obtains pk and computes the RS signature using the Sign oracle provided by $\mathcal{C}^{\mathsf{u}}_{\kappa}$. The remaining simulation is performed as in the original scheme. Eventually, $\mathcal{A}^{\mathsf{gu}}$ outputs a forgery $(\mathsf{m}^\star, \sigma^\star)$ which is opened to signer index u (recall that $\sigma^\star = ((\mathsf{pk}, \sigma_{\mathsf{G}}), (\sigma_{\mathsf{R}}, \mathsf{RED}))$. If u does not exist ($u = \perp$), or u exists and $\mathcal{A}^{\mathsf{gu}}$ neither requested gsk_u nor a group signature on pk for u (i.e., $u \notin \mathcal{Q}^{\mathsf{Key}} \wedge (u, \mathsf{pk}) \notin \mathcal{Q}^{\mathsf{Sign}}$) we abort as we are in the other case. Otherwise, we know that we have a valid RS signature on $\mathsf{m}^\star$ under pk which is not derivable from any queried message (i.e., $\nexists\, (u, \mathsf{m}, \mathsf{ADM}) \in \mathcal{Q}^{\mathsf{Sig}} : \mathsf{m}^\star \overset{\mathsf{ADM}}{\preceq} \mathsf{m}$) and we can output $(\mathsf{m}^\star, \sigma_{\mathsf{R}})$ as an RS forgery.

Overall Bound. The simulations in both reductions are indistinguishable from a real game. In front of an adversary we randomly guess the adversary's strategy, inducing a loss of $1/2$. Our reduction for a Type 1 forger always succeeds if the adversary succeeds, whereas our reduction for Type 2 succeeds with a probability of $1/q$, where $q \leq \mathsf{poly}(\kappa)$ is the number of queries to the Sig oracle. Overall, this means that the probability to break group unforgeability is upper-bounded by $2 \cdot \max\{\varepsilon_{\mathsf{gu}}(\kappa), q \cdot \varepsilon_{\mathsf{p}}(\kappa)\}$. □

Lemma 3. *If Π admits proofs of knowledge, Σ is $\mathsf{EUF\text{-}CMA}$ secure, and Scheme 1 is group unforgeable, then Scheme 1 is also designated-verifier unforgeable.*

Proof. We bound the probability to break designated-verifier unforgeability. We start by defining our opener $\mathsf{O} = (O_1, O_2)$.

$O_1(1^\kappa, n)$: Run $(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}) \leftarrow \mathsf{GS.KeyGen}(1^\kappa, n)$, $(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{E}_1(1^\kappa)$, set $\mathsf{gpk}' \leftarrow (\mathsf{gpk}, \mathsf{crs})$, $\tau' \leftarrow (\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk}, \tau)$ and return $(\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk}, \tau')$.
$O_2(\tau, \mathsf{DVK}, \mathsf{m}^\star, \rho^\star)$: Parse σ as $((\mathsf{pk}, \sigma_{\mathsf{G}}), \overline{\sigma})$ and return $u \leftarrow \mathsf{GS.Open}(\mathsf{gok}, \mathsf{pk}, \sigma_{\mathsf{G}})$.

The tuple $(\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk})$ contained in the output of O_1 is computationally indistinguishable from the output of Setup under the extraction-CRS indistinguishability of the proof system.

What remains is to show that—using O—the success probability of every PPT adversary in the designated-verifier unforgeability game is negligible in the security parameter. We do so by using a sequence of games, where we let $q \leq \mathsf{poly}(\kappa)$ be the number of queries to the DVGen oracle.

Game 0: The original designated-verifier-unforgeability game.
Game 1: As Game 0, but we modify O_1 as follows. We use $\mathcal{C}^{\mathsf{gu}}_{\kappa}$ to denote a group unforgeability challenger.

$O_1(1^\kappa, n)$: Run $\boxed{(\mathsf{gpk}, \mathsf{gok}) \leftarrow \mathcal{C}^{\mathsf{gu}}_{\kappa}}$, set $\boxed{\mathsf{gsk} \leftarrow \perp}$, $(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{E}_1(1^\kappa)$, set $\mathsf{gpk}' \leftarrow (\mathsf{gpk}, \mathsf{crs})$, $\tau' \leftarrow (\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk}, \tau)$ and return $(\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk}, \tau')$.

Then, we simulate all queries to Sig and Key by forwarding them to $\mathcal{C}^{\mathsf{gu}}_{\kappa}$.
Transition - Game 0 → Game 1: This change is conceptual, i.e., $\Pr[S_0] = \Pr[S_1]$.

Game 2: As Game 1, but we guess the index $v^\star$ that the adversary will attack.
Transition - Game 1 → Game 2: The success probability in Game 2 is the same as in Game 1, unless our guess is wrong. That is $\Pr[S_2] = \Pr[S_1] \cdot 1/q$.
Game 3: As Game 2, but in the query to DVGen for user $v^\star$ we engage with an $\mathsf{EUF\text{-}CMA}$ challenger $\mathcal{C}^{\mathsf{f}}_\kappa$, obtain a public key pk and return $\mathsf{vpk}_{v^\star} \leftarrow \mathsf{pk}$. Furthermore, the queries to Sim for user $v^\star$ are simulated without $\mathsf{vsk}_{v^\star}$ by using the Sign oracle provided by $\mathcal{C}^{\mathsf{f}}_\kappa$.
Transition - Game 2 → Game 3: This change is conceptual, i.e., $\Pr[S_3] = \Pr[S_2]$.
Game 4: As Game 3, but for every output of the adversary, we obtain $(\sigma_{\mathsf{R}}, \sigma_{\mathsf{V}}) \leftarrow \Pi.\mathsf{E}_2(\mathsf{crs}, \tau, (\mathsf{m}^\star, \mathsf{pk}, \mathsf{vpk}_{v^\star}), \pi)$. If the extractor fails, we abort.
Transition - Game 3 → Game 4: The success probability in Game 2 is the same as in Game 1, unless the extractor fails, i.e., $|\Pr[S_3] - \Pr[S_4]| \leq \varepsilon_{\mathsf{ext2}}(\kappa)$.

In Game 4 we have two possibilities if $\mathcal{A}$ outputs a valid forgery.

1. We extract a signature σ_{R} such that $\mathsf{RS.Verify}(\mathsf{pk}, \mathsf{m}^\star, \sigma_{\mathsf{R}}) = 1$. Since, our implementation of O_1 does the same as what is done in Open and we have that $(u = \bot \vee (u \notin \mathcal{Q}^{\mathsf{Key}} \wedge \nexists(u, \mathsf{m}, \mathsf{ADM}) \in \mathcal{Q}^{\mathsf{Sig}} : \mathsf{m}^\star \preceq^{\mathsf{ADM}} \mathsf{m}))$ by definition, we can compose $\sigma \leftarrow ((\mathsf{pk}, \sigma_{\mathsf{G}}), (\sigma_{\mathsf{R}}, \bot))$ and return $(\mathsf{m}^\star, \sigma)$ to $\mathcal{C}^{\mathsf{gu}}_\kappa$ as a forgery for the group unforgeability game.
2. We extract a signature σ_{V} such that $\Sigma.\mathsf{Verify}(\mathsf{vpk}_{v^\star}, \mathsf{m}^\star, \sigma_{\mathsf{V}}) = 1$. By definition, we have that $v^\star \notin \mathcal{Q}^{\mathsf{DVKey}} \wedge \nexists(v^\star, \mathsf{m}, \mathsf{ADM}, \cdot, \cdot) \in \mathcal{Q}^{\mathsf{Sim}} : \mathsf{m}^\star \preceq^{\mathsf{ADM}} \mathsf{m})$. Since $\mathsf{m}^\star \preceq^{\mathsf{ADM}} \mathsf{m}$ also includes the identity, i.e., the case where $\mathsf{m}^\star = \mathsf{m}$, we know that $\mathsf{m}^\star$ was never queried to the signing oracle provided by $\mathcal{C}^{\mathsf{f}}_\kappa$ and we can output $(\mathsf{m}^\star, \sigma_{\mathsf{V}})$ as a valid $\mathsf{EUF\text{-}CMA}$ forgery.

The union bound yields $\Pr[S_4] \leq \varepsilon_{\mathsf{gu}}(\kappa) + \varepsilon_{\mathsf{f}}(\kappa)$. Furthermore, we have that $\Pr[S_4] = \Pr[S_3] \cdot (1 - \varepsilon_{\mathsf{ext2}}(\kappa))$, that $\Pr[S_3] = \Pr[S_2] = \Pr[S_1] \cdot 1/q$, and that $\Pr[S_0] = \Pr[S_1]$. All in all this yields $\Pr[S_0] \leq q \cdot (\varepsilon_{\mathsf{gu}}(\kappa) + \varepsilon_{\mathsf{f}}(\kappa) + \varepsilon_{\mathsf{ext2}}(\kappa))$, which proves the lemma. □

Lemma 4. *If Π is witness indistinguishable, then Scheme 1 is simulatable.*

Proof. We show that the output in the simulatability game is (computationally) independent of the bit b.

Game 0: The original simulatability game ($\underline{\sigma}$ is already independent of b).
Game 1: As Game 0, but we obtain crs for the Π upon Setup from a witness indistinguishability challenger $\mathcal{C}^{\mathsf{wi}}_\kappa$ instead of internally generating it.
Transition - Game 0 → Game 1: This change is conceptual, i.e., $\Pr[S_0] = \Pr[S_1]$.
Game 2: As Game 1, but instead of executing Redact inside RoS, we execute the modified algorithm Redact' with additional input vsk_j, which additionally computes $\boxed{\sigma_{\mathsf{V}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{vsk}_j, \mathring{\mathsf{m}})}$ and then computes π as $\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathring{\mathsf{m}}, \mathsf{pk}, \mathsf{vpk}_j), \boxed{(\bot, \sigma_{\mathsf{V}})})$.
Transition - Game 1 → Game 2: A distinguisher $\mathcal{D}^{1\to 2}$ is a distinguisher for adaptive witness indistinguishability of the Π, i.e., $|\Pr[S_3] - \Pr[S_2]| \leq \varepsilon_{\mathsf{wi}}(\kappa)$.

In Game 2, Redact' and Sim are identical, i.e., RoS is independent of b. Thus, the adversary has no advantage in winning the game, i.e., $\Pr[S_2] = 1/2$, which yields $\Pr[S_0] \leq 1/2 + \varepsilon_{\mathsf{wi}}(\kappa)$. □

Lemma 5. *If* GS *is anonymous and* RS *is unforgeable, then Scheme 1 is signer anonymous.*

Proof. We construct an efficient reduction $\mathcal{R}$ which turns an efficient signer-anonymity adversary $\mathcal{A}^{\mathsf{sa}}$ into an efficient adversary $\mathcal{A}$ against anonymity of the underlying GS. $\mathcal{R}$ obtains $(\mathsf{gpk}, \mathsf{gsk})$ from the challenger $\mathcal{C}^{\mathsf{a}}_{\kappa}$ of the anonymity game of GS and completes the setup as in the original scheme. $\mathcal{R}$ simulates the Open oracle by using the Open oracle provided by $\mathcal{C}^{\mathsf{a}}_{\kappa}$ and starts $\mathcal{A}^{\mathsf{sa}}$ on $(\mathsf{gpk}', \mathsf{gsk})$. If $\mathcal{A}^{\mathsf{sa}}$ eventually outputs $b^\star$, then $\mathcal{R}$ outputs $b^\star$ to $\mathcal{C}^{\mathsf{a}}_{\kappa}$. By the RS unforgeability, the simulation of the Open oracle is computationally indistinguishable from a real game. The reduction succeeds with non-negligible probability whenever $\mathcal{A}^{\mathsf{sa}}$ succeeds with non-negligible probability. □

Lemma 6. *If* RS *is private, then Scheme 1 is private.*

Proof. We prove privacy using a sequence of games, where we let $q \leq \mathsf{poly}(\kappa)$ be the number of queries to the Ch oracle.

Game 0: The privacy game with bit $b = 0$.

Game 1_ℓ **(**$1 \leq \ell \leq q$**):** As Game 0, but we set $b = 1$ for the first ℓ queries to Ch.

Transition - Game 0 $\rightarrow$ *Game* 1_1*:* A distinguisher between Game 0 and Game 1_1 is a distinguisher for the RS privacy game. To show this, we engage with an RS privacy challenger $\mathcal{C}^{\mathsf{p}}_{\kappa}$ in the first call to Ch, obtain pk, compute $\sigma_{\mathsf{G}} \leftarrow \mathsf{GS.Sign}(\mathsf{gsk}_i, \mathsf{pk})$, $(\mathring{\mathsf{m}}, \mathring{\sigma}_{\mathsf{R}}) \leftarrow \mathcal{C}^{\mathsf{p}}_{\kappa}.\mathsf{LoRRedact}((\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{ADM}_0), (\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{ADM}_1))$, as well as $\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathring{\mathsf{m}}, \mathsf{pk}, \mathsf{vpk}_j), (\mathring{\sigma}_{\mathsf{R}}, \perp))$, and return $(\mathring{\mathsf{m}}, \underline{\sigma}, \rho) = (\mathsf{m}, (\mathsf{pk}, \sigma_{\mathsf{G}}), ((\mathsf{pk}, \sigma_{\mathsf{G}}), \pi))$. Depending on the bit chosen by $\mathcal{C}^{\mathsf{p}}_{\kappa}$, we either simulate Game 0 or Game 1_1.

Transition - Game $1_\ell \rightarrow 1_{\ell+1}$ $(1 \leq \ell < q)$ *:* The answers of the Ch oracle for the first ℓ queries are already simulated for $b = 1$. As above, a distinguisher between Game 1_ℓ and Game $1_{\ell+1}$ is a RS privacy distinguisher.

In Game 1_q we have a simulation for bit $b = 1$. We can bound probability to distinguish the simulations for $b = 0$ and $b = 1$ by $|\Pr[S_{1_q}] - \Pr[S_0]| \leq q \cdot \varepsilon_{\mathsf{p}}(\kappa)$, which shows that the advantage to win the privacy game is bounded by $1/2 + q \cdot \varepsilon_{\mathsf{p}}(\kappa)$. □

C Proof of Theorem 2

Subsequently, we show that Theorem 2 holds by proving Lemma 7-12.

Lemma 7. *If* GS *is correct,* RS *is correct and adapts signatures,* Σ *is correct, and* Π *is complete, then Scheme 2 is also correct.*

Lemma 7 straight-forwardly follows from inspection; the proof is omitted.

Lemma 8. *If* GS *is traceable and* RS *is unforgeable, then Scheme 2 is group unforgeable.*

Lemma 8 can be proven identically as group unforgeability is proven in the previous section and is therefore omitted.

Lemma 9. *If* Π *is weakly simulation sound extractable,* Σ *is* $\mathsf{EUF\text{-}CMA}$ *secure,* RS *adapts signatures, Scheme 2 is group unforgeable, and the DL assumption holds in* $\mathbb{G}$*, then Scheme 2 is also designated-verifier unforgeable.*

Proof. We subsequently prove designated-verifier unforgeability using a sequence of games. First, we define the opener $\mathsf{O} = (O_1, O_2)$ as follows.

$O_1(1^\kappa, n)$: Run $(\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}) \leftarrow \mathsf{GS.KeyGen}(1^\kappa, n)$, $(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{S}_1(1^\kappa)$, set $\mathsf{gpk}' \leftarrow (\mathsf{gpk}, \mathsf{crs})$, $\tau' \leftarrow (\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}, \tau)$, and return $(\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk}, \tau')$.
$O_2(\tau, \mathtt{DVK}, \mathsf{m}^\star, \rho^\star)$: Parse σ as $((\mathsf{pk}, \sigma_\mathsf{G}), \overline{\sigma})$ and return $u \leftarrow \mathsf{GS.Open}(\mathsf{gok}, \mathsf{pk}, \sigma_\mathsf{G})$.

The tuple $(\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk})$ contained in the output of O_1 is computationally indistinguishable from the output of Setup under the simulation-CRS indistinguishability of the proof system. From now on we will simulate all proofs, i.e., replace all calls to $\Pi.\mathsf{Proof}(\mathsf{crs}, x, w)$ by $\Pi.\mathsf{S}_2(\mathsf{crs}, \tau, x)$.

What remains is to show that—using O—the success probability of every PPT adversary following strategy (1) in the designated-verifier unforgeability game is negligible in the security parameter. We do so by using a sequence of games where we let $q_{\mathsf{Sim}} \leq \mathsf{poly}(\kappa)$ be the number of queries to the Sim oracle and $q \leq \mathsf{poly}(\kappa)$ the number of users in the system.

Game 0: The original designated-verifier unforgeability game.

Game 1: As Game 0, but we modify O_1 as follows, where $\mathcal{C}_\kappa^{\mathsf{gu}}$ denotes a group-unforgeability challenger (note that we can assume that all required accumulator public keys are obtained from collision freeness challengers as all algorithms also run without secret keys without affecting the output distribution of the algorithms):

$O_1(1^\kappa, n)$: Run $\boxed{(\mathsf{gpk}, \mathsf{gok}) \leftarrow \mathcal{C}_\kappa^{\mathsf{gu}}}$, set $\boxed{\mathsf{gsk} \leftarrow \bot}$, $(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{S}_1(1^\kappa)$, $\mathsf{gpk}' \leftarrow (\mathsf{gpk}, \mathsf{crs})$, $\tau' \leftarrow (\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}, \tau)$, and return $(\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk}, \tau')$.

Furthermore, whenever the adversary queries Sig or Key, we use the oracles provided by $\mathcal{C}_\kappa^{\mathsf{gu}}$ to obtain the required group signatures and keys, respectively.

Transition - Game 0 → Game 1: This game change is conceptual and $\Pr[S_1] = \Pr[S_0]$.

Game 2: As Game 1, but whenever the adversary outputs a forgery so that $\mathsf{pk}_\Sigma \cdot \mathsf{pk}'$ corresponds to a key $\mathsf{pk}_\mathsf{R}^\Sigma$ used in a Sim oracle call we check whether the signed accumulator value (used to encode the message) is still the same as the one signed in Sim and abort if so.

Transition - Game 1 → Game 2: If we abort, we have a collision for one of the accumulators. That is, $|\Pr[S_1] - \Pr[S_2]| \leq q_{\mathsf{Sim}} \cdot \varepsilon_{\mathsf{cf}}(\kappa)$.

Game 3: As Game 2, but inside Sim we obtain $\mathsf{pk}_{\mathsf{R}}^{\Sigma}$ from an EUF-CMA challenger of Σ and obtain the required signatures inside $\mathsf{RS.Sign}$ using the Sign oracle provided by the challenger.

Transition - Game 2 → Game 3: This change is conceptual: $\Pr[S_2] = \Pr[S_3]$.[7]

Game 4: As Game 3, but we guess the index $v^\star$ that the adversary will attack. If our guess is wrong, we abort.

Transition - Game 3 → Game 4: The success probability in Game 4 is the same as in Game 3, unless our guess is wrong. That is $\Pr[S_4] = \Pr[S_3] \cdot 1/q$.

Game 5: As Game 4, but in the query to DVGen for user $v^\star$ we engage with an EUF-CMA challenger $\mathcal{C}_\kappa^{\mathsf{f}}$, obtain a public key pk and return $\mathsf{vpk}_{v^\star} \leftarrow \mathsf{pk}$.

Transition - Game 4 → Game 5: This change is conceptual, i.e., $\Pr[S_4] = \Pr[S_5]$.

Game 6: As Game 5, but we further modify O_1 as follows, where $\mathcal{C}_\kappa^{\mathsf{gu}}$ denotes a group-unforgeability challenger:

$O_1(1^\kappa, n)$: Run $(\mathsf{gpk}, \mathsf{gok}) \leftarrow \mathcal{C}_\kappa^{\mathsf{gu}}$, set $\mathsf{gsk} \leftarrow \bot$, $\boxed{(\mathsf{crs}, \tau, \xi) \leftarrow \Pi.\mathsf{S}(1^\kappa)}$, $\mathsf{gpk}' \leftarrow (\mathsf{gpk}, \mathsf{crs})$, $\tau' \leftarrow (\mathsf{gpk}, \mathsf{gok}, \mathsf{gsk}, \tau, \boxed{\xi})$, and return $(\mathsf{gpk}', \mathsf{gok}, \mathsf{gsk}, \tau')$.

Transition - Game 5 → Game 6: This change is conceptual, i.e., $\Pr[S_5] = \Pr[S_6]$.

Game 7: As Game 6, but for every output of the adversary, we check whether pk_{R} corresponds to a key obtained from a challenger in Sim and continue if so. Otherwise, we obtain $(\mathsf{sk}', \mathsf{vsk}_{v^\star}) \leftarrow \Pi.\mathsf{E}(\mathsf{crs}, \xi, (\mathsf{pk}', \mathsf{vpk}_{v^\star}), \pi)$. If the extractor fails, we abort.

Transition - Game 6 → Game 7: Both games proceed identically, unless the extractor fails, i.e., $|\Pr[S_6] - \Pr[S_7]| \leq \varepsilon_{\mathsf{ext}}(\kappa)$.

If the adversary outputs a forgery $(\mathsf{m}^\star, \rho^\star, v^\star)$, where $\rho^\star = (((\mathsf{pk}_\Sigma, \mathsf{pk}_\Lambda), \sigma_\mathsf{G}), \mathsf{pk}', \mathring{\sigma}'_\mathsf{R}, \pi)$, we check whether we have extracted $\mathsf{vsk}_{v^\star}$ such that $\Sigma.\mathsf{VKey}(\mathsf{vpk}_{v^\star}, \mathsf{vsk}_{v^\star}) = 1$. If so, we choose a random message m in the message space of Σ, compute $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{vsk}_{v^\star}, m)$ and output (m, σ) as an EUF-CMA forgery for Σ. Otherwise, we have extracted a secret key sk' such that $\mathsf{RS.VKey}(\mathsf{pk}', \mathsf{sk}') = 1$. Then, we have that $\mathsf{Verify}(\mathsf{gpk}, \mathsf{vpk}_{v^\star}, \mathsf{m}^\star, \rho^\star) = 1$ by definition. If $\mathsf{pk}_\Sigma \cdot \mathsf{pk}'$ corresponds to a key $\mathsf{pk}_\mathsf{R}^\Sigma$ used in Sim, we can output the Σ-signature σ on the accumulator together with the accumulator as an EUF-CMA forgery to one of the challengers from Sim. If not, we can obtain $(\mathsf{pk}, \mathring{\sigma}''_\mathsf{R}) \leftarrow \mathsf{RS.Adapt}(\mathsf{pk}_\Sigma \cdot \mathsf{pk}', \mathsf{m}^\star, \mathring{\sigma}'_\mathsf{R}, -\mathsf{sk}')$ and output $(\mathsf{m}^\star, \sigma) = (\mathsf{m}^\star, (((\mathsf{pk}_\Sigma, \mathsf{pk}_\Lambda), \sigma_\mathsf{G}), (\mathring{\sigma}''_\mathsf{R}, \bot)))$ to break group unforgeability. Note that our implementation of O_2 does the same as what is done in Open and we have that $(u = \bot \ \vee \ (u \notin \mathcal{Q}^{\mathsf{Key}} \ \wedge \ \nexists (u, \mathsf{m}, \mathsf{ADM}) \in \mathcal{Q}^{\mathsf{Sig}} : \mathsf{m}^\star \overset{\mathsf{ADM}}{\preceq} \mathsf{m}))$ by definition. Also note that Game 2 and Game 3 resemble the proof strategy for RS constructions following the paradigm from [DPSS15]. Taking the union bound, the success probability in Game 7 is bounded by $\Pr[S_7] \leq \varepsilon_{\mathsf{gu}}(\kappa) + (1 + q_{\mathsf{Sim}}) \cdot \varepsilon_{\mathsf{f}}(\kappa)$.

[7] Note that the changes in Game 2 and Game 3 resemble the unforgeability proof strategy of [DPSS15, Scheme 1]. For further details, and, in particular for the collision freeness notion of accumulators, see [DPSS15].

Thus, we have that $\Pr[S_0] \leq q \cdot (\varepsilon_{\mathsf{gu}}(\kappa) + (1 + q_{\mathsf{Sim}}) \cdot \varepsilon_{\mathsf{f}}(\kappa) + \varepsilon_{\mathsf{ext}}(\kappa)) + q_{\mathsf{Sim}} \cdot \varepsilon_{\mathsf{cf}}(\kappa)$ which concludes the proof. □

Lemma 10. *If* Π *is witness indistinguishable, and* RS *adapts signatures, then Scheme 2 is simulatable.*

Proof. We prove that the output in the simulatability game is (computationally) independent of the bit b.

Game 0: The original simulatability game ($\underline{\sigma}$ is already independent of b).
Game 1: As Game 0, but we obtain crs for the Π upon Setup from a witness indistinguishability challenger $\mathcal{C}_\kappa^{\mathsf{wi}}$ instead of internally generating it.
Transition - Game 0 → Game 1: This change is conceptual, i.e., $\Pr[S_0] = \Pr[S_1]$.
Game 2: As Game 1, but instead of Redact inside RoS we execute the modified algorithm Redact' which runs on additional input vsk_j and computes π as $\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathsf{pk}', \mathsf{vpk}_j), \boxed{(\bot, \mathsf{vsk}_j)})$.
Transition - Game 1 → Game 2: A distinguisher $\mathcal{D}^{1\to 2}$ is a distinguisher for adaptive witness indistinguishability of Π, i.e., $|\Pr[S_2] - \Pr[S_1]| \leq \varepsilon_{\mathsf{wi}}(\kappa)$.
Game 3: As Game 2, but we further modify Redact' so that it additionally takes ADM as input and works as follows.

$\mathsf{Redact}'(\mathsf{gpk}, \mathsf{vpk}_j, \mathsf{m}, \sigma, \mathsf{MOD}, \mathsf{vsk}_j, \mathsf{ADM})$: Parse σ as $((\mathsf{pk}, \sigma_{\mathsf{G}}), (\sigma_{\mathsf{R}}, \mathsf{RED}))$ and return $(\mathring{\mathsf{m}}, \rho)$, where

$\boxed{\mathsf{sk}_{\mathsf{R}}} \xleftarrow{R} \mathbb{H}, \boxed{\mathsf{pk}_{\mathsf{R}}} \leftarrow \mu(\mathsf{sk}_{\mathsf{R}}), \boxed{\mathsf{pk}' \leftarrow \mathsf{pk}^{-1} \cdot \mu(\mathsf{sk}_{\mathsf{R}})}$
$\boxed{((\mathsf{m}, \sigma'_{\mathsf{R}}), \mathsf{RED}) \leftarrow \mathsf{RS.Sign}((\mathsf{sk}_{\mathsf{R}}, \bot, \mathsf{pk}_\Lambda), \mathsf{m}, \mathsf{ADM})},$
$((\mathring{\mathsf{m}}, \mathring{\sigma}'_{\mathsf{R}}), \cdot) \leftarrow \mathsf{RS.Redact}(\mathsf{pk}_{\mathsf{R}}, \mathsf{m}, \sigma'_{\mathsf{R}}, \mathsf{MOD}, \mathsf{RED}),$
$\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathsf{pk}', \mathsf{vpk}_j), (\bot, \mathsf{vsk}_j))$, and $\rho \leftarrow (\underline{\sigma}, \mathsf{pk}', \mathring{\sigma}'_{\mathsf{R}}, \pi)$.

Transition - Game 2 → Game 3: Under adaptability of the RS, Game 2 and Game 3 are perfectly indistinguishable, i.e., $\Pr[S_3] = \Pr[S_2]$.

In Game 3, Redact' and Sim are identical; RoS is thus independent of b. Thus, the adversary has no advantage in winning the game, i.e., $\Pr[S_3] = 1/2$. Further, we have that $\Pr[S_0] = \Pr[S_1] \leq \Pr[S_2] + \varepsilon_{\mathsf{wi}}(\kappa)$, and that $\Pr[S_3] = \Pr[S_2]$, which yields $\Pr[S_0] \leq 1/2 + \varepsilon_{\mathsf{wi}}(\kappa)$. □

Lemma 11. *If* GS *is anonymous, then Scheme 2 is signer anonymous.*

The proof is identical to the proof of Lemma 5 and therefore not restated here.

Lemma 12. *If* RS *is private and adapts signatures, then Scheme 2 is private.*

Proof. The proof strategy is identical to the privacy proof in the previous section. We however, use the following hybrid to interpolate between the games: We engage with an RS privacy challenger $\mathcal{C}_\kappa^{\mathsf{p}}$ in the $\ell + 1$st call to Ch, obtain pk, compute $\sigma_{\mathsf{G}} \leftarrow \mathsf{GS.Sign}(\mathsf{gsk}_i, \mathsf{pk})$, $(\mathring{\mathsf{m}}, \mathring{\sigma}_{\mathsf{R}}) \leftarrow \mathcal{C}_\kappa^{\mathsf{p}}.\mathsf{LoRRedact}((\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{ADM}_0), (\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{ADM}_1))$, $\mathsf{sk}' \xleftarrow{R} \mathbb{H}$, $\mathsf{pk}' \leftarrow \mu(\mathsf{sk}')$, $(\mathsf{pk}_{\mathsf{R}}, \mathring{\sigma}'_{\mathsf{R}}) \leftarrow \mathsf{RS.Adapt}(\mathsf{pk}, \mathring{\mathsf{m}}, \mathring{\sigma}_{\mathsf{R}}, \mathsf{sk}')$, as well as $\pi \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}, (\mathsf{pk}', \mathsf{vpk}_j), (\mathsf{sk}', \bot))$, and return $(\mathring{\mathsf{m}}, \underline{\sigma}, \rho) = (\mathring{\mathsf{m}}, (\mathsf{pk}, \sigma_{\mathsf{G}}), ((\mathsf{pk}, \sigma_{\mathsf{G}}), \mathsf{pk}', \mathring{\sigma}'_{\mathsf{R}}, \pi))$ Then, depending on the bit chosen by $\mathcal{C}_\kappa^{\mathsf{p}}$, we either simulate Game 0 or Game 1_1 (resp. 1_ℓ or Game $1_{\ell+1}$). □