Selective end-to-end data-sharing in the cloud

Felix Hörandner¹ · Sebastian Ramacher² · Simon Roth³

Received: 27 February 2020 / Accepted: 13 April 2020 / Published online: 2 July 2020
© The Author(s) 2020

Abstract

Cloud-based services, such as Google Drive, Dropbox, or Nextcloud, enable easy-to-use data-sharing between multiple parties, and, therefore, have been widely adopted over the last decade. Nevertheless, privacy challenges hamper their adoption for sensitive data: (1) rather than exposing their private data to a cloud service, users desire end-to-end confidentiality of the shared files without sacrificing usability, e.g., without repeatedly encrypting when sharing the same data set with multiple receivers. (2) Only being able to share full (authenticated) files may force users to expose overmuch information if the data set has not been exactly tailored to the receiver's needs at issue-time. This gap can be bridged by enabling cloud services to selectively disclose only relevant parts of a file without breaking the parts' authenticity. While both challenges have been solved individually, it is not trivial to combine these solutions and maintain their security intentions. In this paper, we tackle this issue and introduce *selective end-to-end data-sharing* by combining ideas from proxy re-encryption (for end-to-end encrypted sharing) and redactable signature schemes (to selectively disclose a subset of still authenticated parts). We overcome the issues encountered when naively combining these two concepts, introduce a security model, and present a modular instantiation together with implementations based on a selection of various building blocks. We give an extensive performance evaluation of our instantiation and conclude with example applications.

Keywords Data-sharing · End-to-end confidentiality · Proxy re-encryption · Redactable signatures

1 Introduction

The advancement of cloud-based infrastructure enabled many new applications. One prime example is the vast landscape of cloud storage providers, such as Google, Apple, Microsoft, and others, but also including many solutions for federated cloud storage, such as Nextcloud. All of them offer the same essential and convenient-to-use functionality: users upload files and can later share these files on demand with others on a per-file basis or a more coarse level of granularity. Of course, when sharing sensitive data (e.g., medical records), the intermediate cloud storage provider needs to be

highly trusted to operate on plain data, or a protection layer is required to ensure the *end-to-end confidentiality* between users of the system. Additionally, many use cases rely on the *authenticity* of the shared data. However, if the authenticated file was not explicitly tailored to the final receivers, e.g., because the receivers were yet unknown at the issuing time, or because they have conflicting information requirements, users are forced to expose additional unneeded parts contained in the authenticated file to satisfy the receivers' needs. Such a mismatch in the amount of issued data and data required for a use case can prevent adoption not only by privacy-conscious users but also due to legal requirements (c.f. EU's GDPR [23]). To overcome this mismatch, the cloud system should additionally support convenient and efficient *selective disclosure* of data to share specific parts of a document depending on the receiver. E.g., even if a doctor only issues a single document, the patient would be able to selectively share the parts relevant to the doctor's prescribed absence with an employer, other parts on the treatment cost with an insurance, and again different parts detailing the diagnosis with a specialist for further treatment. Therefore, we aim to combine end-to-end confidentiality and selective dis-

✉ Felix Hörandner
felix.hoerandner@iaik.tugraz.at

Sebastian Ramacher
sebastian.ramacher@ait.ac.at

Simon Roth
simon.roth@prime-sign.com

¹ Graz University of Technology, Graz, Austria

² AIT Austrian Institute of Technology, Vienna, Austria

³ PrimeSign GmbH, Graz, Austria

closure of authentic data to what we call *selective end-to-end data-sharing*.

End-to-end confidentiality In the cloud-based document sharing setting, the naïve solution employing public-key encryption has its fair share of drawbacks. While such an approach would work for users to outsource data storage, it falls flat as soon as users desire to share files with many users. In a naïve approach based on public-key encryption, the sender would have to encrypt the data (or in a hybrid setting, the symmetric keys) separately for each receiver, which would require the sender to fetch the data from cloud storage, encrypt them locally, and upload the new ciphertext again and again. Proxy re-encryption (PRE), envisioned by Blaze et al. [5] and later formalized by Ateniese et al. [2], solves this issue conveniently: Users initially encrypt data to themselves. Once they want to share that data with other users, they provide a re-encryption key to a so-called proxy, which is then able to transform the ciphertext into a ciphertext for the desired receiver, without ever learning the underlying message. Finally, the receiver downloads the re-encrypted data and decrypts them with her key. The re-encryption keys can be computed non-interactively, i.e., without the receiver's involvement. Also, proxy re-encryption gives the user the flexibility to not only forward ciphertexts after they were uploaded, but also to generate re-encryption keys to enable sharing of data that will be uploaded in the future. However, note that by employing proxy re-encryption, we still require the proxy to execute the re-encryption algorithms honestly. More importantly, it is paramount that the proxy server securely handles the re-encryption keys. While the re-encryption keys generated by a sender are not powerful enough to decrypt a ciphertext on their own, combined with the secret key of a receiver, any ciphertext of the sender could be re-encrypted and, finally, decrypted.

Authenticity and selective disclosure Authenticity of the data is easily achievable for the full data stored in a file. Ideally, the issuer generates a signature only over the minimal subset of data that is later required by the receiver for a given use case. Unfortunately, the issuer would need to know all relevant use cases in advance at sign time to create appropriate signed documents for each case or later interactively re-create signatures over specified sub-sets on demand. The problem becomes more interesting when one of the desired features involves selectively disclosing only parts of an authenticated file. Naively, one could authenticate the parts of a file separately and then disclose individual parts. However, at that point, one loses the link between the parts and the other parts of the complete file. More sophisticated approaches have been proposed over the last decades, for example, based on Merkle trees, which we summarize for this work as redactable signature schemes (RSS) [27]. With RSS, starting from a signature on a file, anyone can repeatedly redact the signed

message and update the signature accordingly to obtain a resulting signature-message pair that only discloses a desired subset of parts. Thereby it is guaranteed, that the redacted signature was produced from the original message and the signature does not leak the redacted parts.

Applications of selective end-to-end data-sharing Besides the use in e-health scenarios, which we use throughout this article as an example, we believe this concept also holds value for a broader set of use cases, wherever users want to share privacy-relevant data between two domains that produce and consume different sets of data. A short selection is introduced below: (1) *Expenses* To get a refund for travel expenses, an employee selectively discloses relevant items on her bank-signed credit card bill, without exposing unrelated payments which may reveal privacy-sensitive information. (2) *Commerce* Given a customer-signed sales contract, an online marketplace wants to comply with the GDPR and preserve its customers' privacy while using subcontractors. The marketplace redacts the customer's name and address but reveals product and quantity to its supplier, and redacts the product description but reveals the address to a delivery company. (3) *Financial statements* A user wants to prove eligibility for a discount/service by disclosing her income category contained in a signed tax document, without revealing other tax-related details such as marriage status, donations, income sources, etc. Similarly, a user may need to disclose the salary to a future landlord, while retaining the secrecy of other details. (4) *Businesses* Businesses may employ selective end-to-end data-sharing to securely outsource data storage and sharing to a cloud service in compliance with the law. To honor the users' right to be forgotten, the company could order the external storage provider to redact all parts about that user, rather than to download, to remove, to re-sign, and to upload the file again. (5) *Identity management* Given a government-issued identity document, users instruct their identity provider (acting as cloud storage) to selectively disclose the minimal required set of contained attributes for the receiving service providers. In this use case, unlinkability might also be desired.

Contribution We propose *selective end-to-end data-sharing* for cloud systems to provide end-to-end confidentiality, authenticity, and selective disclosure. This article extends upon our conference paper [26].

Firstly, we formalize an *abstract model* and its associated security properties. At the time of encrypting or signing, the final receiver or the required minimal combination of parts, respectively, might not yet be known. Therefore, our model needs to support ad-hoc and selective sharing of protected files or their parts. Besides the data *owner*, we require distinct *senders*, who can encrypt data for the owner, as well as *issuers* that certify the authenticity of the data. Apart from *unforgeability*, we aim to conceal the plain text from unau-

thorized entities, such as the proxy (i.e. *proxy privacy*), and additionally information about redacted parts from receivers (i.e. *receiver privacy*). Further, we define *transparency* to hide whether a redaction happened or not.

Secondly, we present a *modular construction* for our model by using cryptographic building blocks that can be instantiated with various schemes, which enables to tailor this construction to specific applications' needs. A challenge for combining was that RSS generally have access to the plain message in the redaction process to update the signature. However, we must not expose the plain message to the proxy. Even if black-box redactions were possible, the signature must not leak any information about the plaintext, which is not a well-studied property in the context of RSS. We avoid these problems by signing symmetric encryptions of the message parts. To ensure that the signature corresponds to the original message and no other possible decryptions, we generate a commitment on the used symmetric key and add this commitment as another part to the redactable signature.

Thirdly, we *evaluate three implementations* of our modular construction that are built on different underlying primitives, namely two implementations with RSS for unordered data with CL [10] and DHS [17,18] accumulators, as well as an implementation supporting ordered data. To give an impression for real-world usage, we perform the benchmarks with various combinations of part numbers and sizes, on a PC, server, and phone.

In addition to the conference version [26], we illustrate *example applications* within the banking context. Additionally, we propose an encoding layer to transform ordered and hierarchical data structures, such as the widely adopted JSON format, into a set of message parts that can be used with our selective end-to-end data-sharing mechanism.

2 Related work

Proxy re-encryption introduced by Blaze et al. [5], enables a semi-trusted proxy to transform ciphertext for one user into ciphertext of the same underlying message now encrypted for another user, where the proxy does not learn anything about the plain message. Ateniese et al. [2] proposed the first strongly secure constructions, while follow-up work focused on stronger security notions [11,30], performance improvements [15], and features such as forward secrecy [19], key-privacy [1], or lattice-based instantiations [12].

Attribute-based encryption (ABE) [24,34,35] is a well-known primitive enabling fine-grained access to encrypted data. The idea is, that a central authority issues private keys that can be used to decrypt ciphertexts depending on attributes and policies. While ABE enables this fine-grained access control based on attributes, it is still all-or-nothing concerning encrypted data.

Functional encryption (FE) [6], a generalization of ABEs, enables to define functions on the encrypted plaintext given specialized private keys. To selectively share data, one could distribute the corresponding private keys where the functions only reveal parts of the encrypted data. Consequently, the ability to selectively share per ciphertext is lost without creating new key pairs.

Redactable signatures [27,37] enable to redact (i.e., black-out) parts of a signed document that should not be revealed, while the authenticity of the remaining parts can still be verified. This concept was extended for specific data structures such as trees [7,36] and graphs [29]. As a stronger privacy notion, transparency [7] was added to capture if a scheme hides whether a redaction happened or not. A generalized framework for RSS is offered by Derler et al. [21]. Further enhancements enable only a designated verifier, but not a data thief, to verify the signature's authenticity and anonymize the signer's identity to reduce metadata leakage [20]. We refer to [16] for a comprehensive overview.

Homomorphic signatures [27] make it possible to evaluate a function on the message-signature pair where the outcome remains valid. Such a function can also be designed to remove (i.e., redact) parts of the message and signature. The concept of homomorphic proxy re-authenticators [22] applies proxy re-encryption to securely share and aggregate such homomorphic signatures in a multi-user setting. However, homomorphic signatures do not inherently provide support for defining which redactions are admissible or the notion of transparency.

Attribute-based credentials (ABCs or anonymous credentials) enable to only reveal a minimal subset of authentic data. In such a system, an issuer certifies information about the user as an anonymous credential. The user may then compute presentations containing the minimal data set required by a receiver, which can verify the authenticity. Additionally, ABCs offer unlinkability, i.e., they guarantee that no two actions can be linked by colluding service providers and issuers. This concept was introduced by Chaum [13,14] and the most prominent instantiations are Microsoft's U-Prove [33] and IBM's identity mixer [8,9]. However, as the plain data is required to compute the presentations, this operation must be performed in a sufficiently trusted environment. In previous work [25], the need for selective disclosure was identified in a semi-trusted cloud environment and informally proposed to combine PRE and RSS, but did not yet provide a formal definition or concrete constructions.

The cloudification of ABCs [28] represents the closest related research to our work and to filling this gap. Their concept enables a semi-trusted cloud service to derive representations from encrypted credentials without learning the underlying plaintext. Also, unlinkability is further guaranteed protecting the users' privacy, which makes it a very good choice for identity management where only small amounts of identity

attributes are exchanged. However, this property becomes impractical with larger documents as hybrid encryption trivially breaks unlinkability. In contrast, our work focuses on a more general model with a construction that is efficient for both small as well as large documents. In particular, our construction (1) already integrates hybrid encryption for large documents avoiding ambiguity of the actually signed content, and (2) supports features of redactable signatures such as the transparency notion, signer-defined admissible redactions, as well as different data structures. These features come at a cost: the proposed construction for our model does not provide unlinkability.

3 Preliminaries

This section recalls the definitions of our cryptographic building blocks, namely proxy re-encryption and redactable signatures.

Proxy re-encryption We recall the syntax of proxy re-encryption (PRE) [2,5,30], where we focus on single-hop, non-interactive, unidirectional proxy re-encryption schemes. For the security definitions, we refer to Appendix A.1.

Definition 1 (PRE) A proxy re-encryption (PRE) scheme over message space $\mathcal{M}$ consists of the PPT algorithms (Setup, KeyGen, **Enc**, **Dec**, ReKeyGen, ReEnc) with **Enc** = $(\text{Enc}^j)_{j \in [2]}$ and **Dec** = $(\text{Dec}^j)_{j \in [2]}$. For $j \in [2]$, they are defined as follows.

$\text{Setup}(1^\kappa) \rightarrow \text{pp}$: On input of a security parameter κ , outputs public parameters pp .

$\text{KeyGen}(\text{pp}) \rightarrow (\text{sk}, \text{pk})$: On input of a public parameters pp , outputs a secret and public key (sk, pk) .

$\text{Enc}^j(\text{pk}, m) \rightarrow c$: On input of a public key pk , and a message $m \in \mathcal{M}$ outputs a level j ciphertext c .

$\text{Dec}^j(\text{sk}, c) \rightarrow m$: On input of a secret key sk , and level j ciphertext c , outputs $m \in \mathcal{M} \cup \{\perp\}$.

$\text{ReKeyGen}(\text{sk}_A, \text{pk}_B) \rightarrow \text{rk}_{A \rightarrow B}$: On input of a secret key sk_A for A and a public key pk_B for B , outputs a re-encryption key $\text{rk}_{A \rightarrow B}$.

$\text{ReEnc}(\text{rk}_{A \rightarrow B}, c_A) \rightarrow c_B$: On input of a re-encryption key $\text{rk}_{A \rightarrow B}$, and a ciphertext c_A for user A , outputs a ciphertext c_B for user B .

Redactable signatures We recall the generalized model for redactable signatures from [21], which builds on [7]. By ADM we denote a data structure encoding the admissible redactions of some message m . The modification instructions for some message are denoted as MOD. $\text{MOD} \stackrel{\text{ADM}}{\preceq} m$ means that MOD is a valid redaction description with respect to ADM and m . $\text{ADM} \preceq m$ denotes that ADM matches m . By $m' \stackrel{\text{MOD}}{\leftarrow} m$, we

Fig. 1 System context

denote the derivation of m' from m with respect to MOD. For the security definitions, we refer to Appendix A.2.

Definition 2 (Redactable signature scheme) A redactable signature scheme (RSS) is a tuple of algorithms (KeyGen, Sign, Verify, Redact) of PPT algorithms, which are defined as follows:

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

$\text{Sign}(\text{sk}, m, \text{ADM}) \rightarrow (\sigma, \text{RED})$: Takes a secret key sk , a message m , and admissible modifications ADM as input, and outputs a signature σ together with some auxiliary redaction information RED.¹

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

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

4 System context

This section gives an abstract description of the system's actors, their interactions, and their trust relationships.

Actors and data flow As an informal overview, Fig. 1 illustrates the system model in the context of interactions between the following five actors: (1) the *issuer* signs the plain data. For example, a hospital or government agency may certify the data owner's health record or identity data, respectively. (2) The *sender* encrypts the signed data for the data owner. (3) The *data owner* is the entity, for which the data was originally encrypted. Initially, only this owner can decrypt the ciphertext. The owner may decide to selectively share data with other receivers. In our model, the data owner specifies

¹ We assume that ADM can always be correctly derived from any valid message-signature pair.

Fig. 2 Algorithms performed by actors (boxes denote trust boundaries)

which ciphertext parts to redact and generates re-encryption keys to delegate decryption rights to other receivers. (4) The *proxy* redacts specified parts of a signed and encrypted message. Then, the proxy uses a re-encryption key to transform the remaining parts, which are encrypted for one entity (the owner), into ciphertext for another entity (a receiver). (5) Finally, the *receiver* is able to decrypt the non-redacted parts and verify their authenticity. Multiple of these roles can be held by the same entity. For example, a data owner signs her data (as issuer), uploads the data (as sender), or accesses the previously data (as receiver).

Trust The interactions between the above-presented actors are carried out in the context of the following trust relationships and assumptions, which also shape the design of our proposed model. (1) The data owner assumes that the proxy is honest but curious, i.e., that the proxy performs the protocol correctly, but tries to learn information about the user. Consequently, an objective of our model is to protect the user's privacy against the proxy. (2) The data owner trusts the receiver to see some of her data, but not everything. Our model features selective disclosure, which enables to only reveal parts of a file to the receiver. (3) The proxy and receivers selected by a data owner must not collude. Otherwise, the proxy could share the user's full data with the (corrupted) receiver, who is able to decrypt it. Technically, the proxy would re-encrypt the user's data with the user-supplied re-encryption key towards the corrupted receiver, who would decrypt the data with its private key. This criterion has to be considered by the user when selecting receivers and proxies. (4) Receivers need to be able to trust in received data of/about the user. Without sufficient trust between user and receiver, self-claimed data cannot be accepted. Instead, the receiver may require an authenticity guarantee by an issuer, who is trusted by both data owner and receiver w.r.t. the certified data.

5 Definition: selective end-to-end data-sharing

We present an abstract model for selective end-to-end data-sharing and define security properties. It is our goal to formalize a generic framework that enables various instantiations.

5.1 Definition

Figure 2 gives an overview of the model's algorithms, their allocation at different actors, and their interactions. In our following definitions, we adapt the syntax and notions inspired by standard definitions of PRE [2] and RSS [21].

Definition 3 (Selective End-to-End Data-Sharing) A scheme for selective end-to-end data-sharing (SEEDS) consists of the PPT algorithms as defined below. The algorithms return an error symbol $\perp$ if their input is not consistent.

SignKeyGen(1^κ) $\rightarrow$ (ssk , spk): On input of a security parameter κ , this probabilistic algorithm outputs a signature keypair (ssk , spk).

EncKeyGen(1^κ) $\rightarrow$ (esk , epk): On input of a security parameter κ , this probabilistic algorithm outputs an encryption keypair (esk , epk).

ReKeyGen(esk_A , epk_B) $\rightarrow$ $rk_{A \rightarrow B}$: On input of a private encryption key esk_A of user A , and a public encryption key epk_B of user B , this (probabilistic) algorithm outputs a re-encryption key $rk_{A \rightarrow B}$.

Sign(ssk , m , ADM) $\rightarrow$ σ : On input of a private signature key ssk , a message m and a description of admissible messages ADM , this (probabilistic) algorithm outputs the signature σ . The admissible redactions ADM specifies which message parts must not be redacted.

$\text{Verify}(\text{spk}, m, \sigma) \rightarrow \text{valid}$: On input of a public key spk , a signature σ and a message m , this deterministic algorithm outputs a bit $\text{valid} \in \{0, 1\}$.

$\text{Encrypt}(\text{epk}_A, m, \sigma) \rightarrow c_A$: On input of a public encryption key epk_A , a message m and a signature σ , this (probabilistic) algorithm outputs the ciphertext c_A .

$\text{Decrypt}(\text{esk}_A, c_A) \rightarrow (m, \sigma)$: On input of a private decryption key esk_A , a signed ciphertext c_A , this deterministic algorithm outputs the underlying plain message m and signature σ if the signature is valid, and $\perp$ otherwise.

$\text{Redact}(c_A, \text{MOD}) \rightarrow c'_A$: This (probabilistic) algorithm takes a valid, signed ciphertext c_A and modification instructions MOD as input. MOD specifies which message parts should be redacted. The algorithm returns a redacted signed ciphertext c'_A .

$\text{ReEncrypt}(\text{rk}_{A \rightarrow B}, c_A) \rightarrow c_B$: On input of a re-encryption key $\text{rk}_{A \rightarrow B}$ and a signed ciphertext c_A , this (probabilistic) algorithm returns a transformed ciphertext c_B of the same message.

5.2 Correctness

Informally, the correctness property requires that all honestly signed, encrypted, and possibly redacted and re-encrypted ciphertexts can be correctly decrypted and verified. Formally:

$$\begin{aligned} & \forall k \in N, \forall (\text{ssk}, \text{spk}) \leftarrow \text{SignKeyGen}(1^k) \\ & \quad \forall (\text{esk}_A, \text{epk}_A) \leftarrow \text{EncKeyGen}(1^k) \\ & \quad \forall (\text{esk}_B, \text{epk}_B) \leftarrow \text{EncKeyGen}(1^k) \\ & \quad \forall \text{rk}_{A \rightarrow B} \leftarrow \text{ReKeyGen}(\text{esk}_A, \text{epk}_B) \\ & \forall m, \forall \text{ADM} \preceq m, \forall \sigma \leftarrow \text{Sign}(\text{ssk}, m, \text{ADM}) \\ & \quad \forall c_A \leftarrow \text{Encrypt}(\text{epk}_A, m, \sigma) \\ & \quad \forall c_B \leftarrow \text{ReEncrypt}(\text{rk}_{A \rightarrow B}, c_A), \\ & \quad \forall \text{MOD} \preceq^{\text{ADM}} m, \forall c'_A \leftarrow \text{Redact}(c_A, \text{MOD}) \\ & \quad \forall c'_B \leftarrow \text{ReEncrypt}(\text{rk}_{A \rightarrow B}, c'_A), \end{aligned}$$

it holds that

$$\begin{aligned} & \text{Decrypt}(\text{esk}_A, c_A) = \text{Decrypt}(\text{esk}_B, c_B) = (m, \sigma), \\ & \text{Verify}(\text{spk}, m, \sigma) = 1, \\ & (m', \sigma') \leftarrow \text{Decrypt}(\text{esk}_A, c'_A), \text{Verify}(\text{spk}, m', \sigma') = 1, \\ & (m', \sigma'') \leftarrow \text{Decrypt}(\text{esk}_B, c'_B), \text{Verify}(\text{spk}, m', \sigma'') = 1, \\ & \text{with } m' \xleftarrow{\text{MOD}} m_i. \end{aligned}$$

5.3 Oracles

To keep the our security experiments concise, we define various oracles. The adversaries are given access to a subset

of these oracles in the security experiments. These oracles are implicitly able to access public parameters and keys generated in the security games. The environment maintains the following initially empty sets: HU for honest users, CU for corrupted users, CH for challenger users, SK for signature keys, EK for encryption keys, and Sigs for signatures.

$\text{AU}(i, t)$: The add user oracle generates users and tracks their key pairs.

if $i \in \text{HU} \cup \text{CU} \cup \text{CH}$ or $t = \text{CH}$: **return** $\perp$
 add i to set of type t
 $\text{EK}[i] \leftarrow (\text{esk}_i, \text{epk}_i) \leftarrow \text{EncKeyGen}(1^k)$
 $\text{SK}[i] \leftarrow (\text{ssk}_i, \text{spk}_i) \leftarrow \text{SignKeyGen}(1^k)$
if $t = \text{CU}$: **return** $((\text{esk}_i, \text{epk}_i), (\text{ssk}_i, \text{spk}_i))$
if $t = \text{HU}$: **return** $((\perp, \text{epk}_i), (\perp, \text{spk}_i))$

$\text{SIG}(i, m, \text{ADM})$: The sign oracle signs messages for the challenge and honest users, and tracks all signatures.

if $i \notin \text{CH} \cup \text{HU}$ or $\text{SK}[i][0] = \perp$: **return** $\perp$
 $\sigma \leftarrow \text{Sign}(\text{SK}[i][0], m, \text{ADM})$
 $\text{Sigs} \leftarrow \text{Sigs} \cup \{m' \mid \forall \text{MOD} \preceq^{\text{ADM}} m \forall m' \xleftarrow{\text{MOD}} m\}$
return (m, σ)

$\text{SE}(i, j, m, \text{ADM})$: The sign-encrypt oracle signs and encrypts messages for any user, and tracks all signatures.

if $i, j \notin \text{CH} \cup \text{HU} \cup \text{CU}$: **return** $\perp$
 $\sigma \leftarrow \text{Sign}(\text{SK}[i][0], m, \text{ADM})$
 $c_A \leftarrow \text{Encrypt}(\text{EK}[j][1], m, \sigma)$
 $\text{Sigs} \leftarrow \text{Sigs} \cup \{m' \mid \forall \text{MOD} \preceq^{\text{ADM}} m \forall m' \xleftarrow{\text{MOD}} m\}$
return c_A

$\text{RKG}(i, j)$: The re-encryption key generator oracle generates re-encryption keys except for transformation from the challenge user to a corrupted user.

if $i \in \text{CH}$ and $j \in \text{CU}$: **return** $\perp$
if $i, j \notin \text{HU} \cup \text{CU} \cup \text{CH}$ or $i = j$: **return** $\perp$
return $\text{ReKeyGen}(\text{EK}[i][0], \text{EK}[j][1])$

$\text{RE}(i, j, c_i)$: The re-encryption oracle re-encrypts submitted ciphertexts as long as the target user is not corrupted and the ciphertexts are not derived from the challenge.

if $i, j \notin \text{HU} \cup \text{CU} \cup \text{CH}$ or $i = j$: **return** $\perp$
if c_i is not a proper 2nd-level ciphertext for $\text{EK}[i][0]$: **return** $\perp$
if 1) the oracle is called in the guess phase, and 2) $j \in \text{CU}$, and 3) c_i is a derivative of C^* , that is if $(m, \sigma) \leftarrow \text{Decrypt}(\text{EK}[i][0], c_i)$ and $m \subseteq m_0$ or $m \subseteq m_1$: **return** $\perp$
 $\text{rk}_{i \rightarrow j} \leftarrow \text{ReKeyGen}(\text{EK}[i][0], \text{EK}[j][1])$
return $\text{ReEncrypt}(\text{rk}_{i \rightarrow j}, c_i)$.

$\text{D}(i, c_i)$: The decrypt oracle decrypts ciphertexts as long as they are not derived from the challenge ciphertext.

```

if  $i \notin \text{HU} \cup \text{CU} \cup \text{CH}$ : return  $\perp$ 
if  $c_i$  is not a proper ciphertext for  $\text{EK}[i][0]$ :
  return  $\perp$ 
if 1) the oracle is called in the guess phase, and
  2)  $c_i$  is a derivative of  $C^*$ , that is if  $(m, \sigma) \leftarrow \text{Decrypt}(\text{EK}[i][0], c_i)$  and  $m \subseteq m_0$  or  $m \subseteq m_1$ :
    return  $\perp$ 
return  $(m, \sigma) \leftarrow \text{Decrypt}(\text{EK}[i][0], c_i)$ 

```

5.4 Security notions

Unforgeability Unforgeability requires that it should be infeasible to compute a valid signature σ for a given public key spk on a message m without knowledge of the corresponding signing key ssk . The adversary may obtain signatures of other users and therefore is given access to a signing oracle (SIG). Of course, we exclude signatures or their redactions that were obtained by adaptive queries to that signature oracle.

Experiment $\text{Exp}_{\text{SEEDS}, \mathcal{A}}^{\text{Unf}}(1^\kappa)$

```

CH  $\leftarrow \{0\}$ ,  $\text{SK}[0] \leftarrow (\text{ssk}^*, \text{spk}^*) \leftarrow \text{SignKeyGen}(1^\kappa)$ 
 $\mathcal{O} \leftarrow \{\text{SIG}\}$ ,  $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(\text{spk}^*)$ 
if  $\text{Verify}(\text{spk}^*, m, \sigma) = 1$  and  $m \notin \text{Sigs}$ : return 1
else: return 0

```

Experiment 1: Unforgeability Experiment for Signatures of SEEDS Schemes

Definition 4 (Unforgeability) A SEEDS scheme is unforgeable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr \left[\text{Exp}_{\text{SEEDS}, \mathcal{A}}^{\text{Unf}}(1^\kappa) = 1 \right] < \varepsilon(\kappa)$$

Proxy privacy Proxy privacy captures that proxies should not learn anything about the plain data of ciphertext while processing them with the Redact and ReEncrypt operations. This property is modeled as an IND-CCA style game, where the adversary is challenged on a signed and encrypted message. Since the proxy may learn additional information in normal operation, the adversary gets access to several oracles: Obtaining additional ciphertexts is modeled with a Sign-and-Encrypt oracle (SE). A proxy would also get re-encryption keys enabling re-encryption operations between corrupt and honest users (RE, RKG). Furthermore, the adversary even gets a decryption oracle (D) to capture that the proxy colludes with a corrupted receiver, who reveals the plaintext of ciphertext processed by the proxy. We exclude operations that would trivially break the game, such as re-encryption keys from the challenge user to a corrupt user, or re-encryptions and decryptions of (redacted) ciphertexts of the challenge.

Experiment $\text{Exp}_{\text{SEEDS}, \mathcal{A}}^{\text{PP}}(1^\kappa)$

```

 $\text{SK}[0] \leftarrow (\text{ssk}^*, \text{spk}^*) \leftarrow \text{SignKeyGen}(1^\kappa)$ 
 $\text{EK}[0] \leftarrow (\text{esk}^*, \text{epk}^*) \leftarrow \text{EncKeyGen}(1^\kappa)$ 
CH  $\leftarrow \{0\}$ ,  $b \xleftarrow{R} \{0, 1\}$ 
 $\mathcal{O}_1 \leftarrow \{\text{AU}, \text{SE}, \text{RKG}, \text{RE}, \text{D}\}$ ,  $\mathcal{O}_2 \leftarrow \{\text{SE}, \text{RE}, \text{D}\}$ 
 $(m_0, \text{ADM}_0, m_1, \text{ADM}_1, \text{st}) \leftarrow \mathcal{A}^{\mathcal{O}_1}(\text{spk}^*, \text{epk}^*)$ 
if  $m_0 \approx m_1$ , or  $\text{ADM}_0 \approx \text{ADM}_1$ : abort
 $\sigma \leftarrow \text{Sign}(\text{ssk}^*, m_b, \text{ADM}_b)$ 
 $c^* \leftarrow \text{Encrypt}(\text{epk}^*, m_b, \sigma)$ 
 $b' \leftarrow \mathcal{A}^{\mathcal{O}_2}(\text{st}, c^*)$ 
if  $b = b'$ : return 1, else: return 0

```

Experiment 2: Proxy Privacy Experiment for Ciphertexts of SEEDS Schemes ($X \sim Y \dots |X| = |Y|$ and corresponding items x_i, y_i have same length)

Definition 5 (Proxy privacy) A SEEDS scheme is proxy private, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\left| \Pr \left[\text{Exp}_{\text{SEEDS}, \mathcal{A}}^{\text{PP}}(1^\kappa) = 1 \right] - 1/2 \right| < \varepsilon(\kappa)$$

Experiment $\text{Exp}_{\text{SEEDS}, \mathcal{A}}^{\text{RP}}(1^\kappa)$

```

 $\text{SK}[0] \leftarrow (\text{ssk}^*, \text{spk}^*) \leftarrow \text{SignKeyGen}(1^\kappa)$ 
 $\text{EK}[0] \leftarrow (\text{esk}^*, \text{epk}^*) \leftarrow \text{EncKeyGen}(1^\kappa)$ 
CH  $\leftarrow \{0\}$ ,  $b \xleftarrow{R} \{0, 1\}$ 
 $\mathcal{O} \leftarrow \{\text{AU}, \text{SIG}, \text{LoRRedact}(\dots, b)\}$ 
 $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\text{spk}^*, \text{epk}^*)$ 
if  $b = b'$ : return 1, else: return 0
with  $\text{LoRRedact}(i, j, m_0, \text{MOD}_0, m_1, \text{MOD}_1, \text{ADM}, b)$ :
if  $i \notin \text{CH}$  or  $j \notin \text{HU} \cup \text{CU}$ : return  $\perp$ 
 $\text{rk}_{A \rightarrow B} \leftarrow \text{ReKeyGen}(\text{EK}[i][0], \text{EK}[j][1])$ 
for both  $c \in \{0, 1\}$ 
   $\sigma_c \leftarrow \text{Sign}(\text{SK}[i][0], m_c, \text{ADM})$ 
   $c_{A,c} \leftarrow \text{Encrypt}(\text{EK}[i][1], m_c, \sigma_c)$ 
   $c'_{A,c} \leftarrow \text{Redact}(c_{A,c}, \text{MOD}_c)$ 
   $c'_{B,c} \leftarrow \text{ReEncrypt}(\text{rk}_{A \rightarrow B}, c'_{A,c})$ 
   $(m'_c, \sigma'_c) \leftarrow \text{Decrypt}(\text{EK}[j][0], c'_{B,c})$ 
if  $m'_0 = m'_1$ : return  $c'_{B,b}$ , else: return  $\perp$ 

```

Experiment 3: Receiver Privacy Experiment for SEEDS Schemes

Receiver privacy Receiver privacy captures that users only want to share information selectively. Therefore, receivers should not learn any information on parts that were redacted when given a redacted ciphertext. Since receivers may additionally obtain decrypted messages and their signatures during normal operation, the adversary gets access to a signature oracle (SIG). The experiment relies on another oracle (LoRRedact), that simulates the proxy's output. One of two messages is chosen with challenge bit b , redacted, re-encrypted and returned to the adversary to guess b . To avoid trivial attacks, the remaining message parts must be a valid subset of the other message's parts. If the ciphertext leaks information about the redacted parts, the adversary could exploit this to win.

Definition 6 (Receiver privacy) A SEEDS scheme is receiver private, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\left| \Pr \left[\text{Exp}_{\text{SEEDS}, \mathcal{A}}^{\text{RP}}(1^\kappa) = 1 \right] - 1/2 \right| < \varepsilon(\kappa)$$

Transparency Additionally, a SEEDS scheme may provide transparency. For example, considering a medical report, privacy alone might hide what treatment a patient received, but not the fact that some treatment was administered. Therefore, it should be infeasible to decide whether parts of an encrypted message were redacted or not. Again, the adversary gets access to a signature oracle (SIG) to cover the decrypted signature message pairs during normal operation of receivers. The experiment relies on another oracle (RedactOrNot), that simulates the proxy's output. Depending on the challenge bit b , the adversary gets a ciphertext that was redacted or a ciphertext over the same subset of message parts generated through the sign operation but without redaction. If the returned ciphertext leaks information about the fact that redaction was performed or not, the adversary could exploit this to win.

Experiment $\text{Exp}_{\text{SEEDS}, \mathcal{A}}^{\text{Trans}}(1^\kappa)$

$\text{SK}[0] \leftarrow (\text{ssk}^*, \text{spk}^*) \leftarrow \text{SignKeyGen}(1^\kappa)$
 $\text{EK}[0] \leftarrow (\text{esk}^*, \text{epk}^*) \leftarrow \text{EncKeyGen}(1^\kappa)$
 $\text{CH} \leftarrow \{0\}, b \xleftarrow{R} \{0, 1\}$
 $\mathcal{O} \leftarrow \{\text{AU}, \text{SIG}, \text{RedactOrNot}(\cdot, \cdot, \cdot, \cdot, b)\}$
 $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\text{esk}^*, \text{epk}^*, \text{spk}^*)$
if $b = b'$: **return** 1, **else**: **return** 0

with $\text{RedactOrNot}(i, m, \text{MOD}, \text{ADM}, b)$:

if $i \notin \text{CH}$: **return** $\perp$
 $\sigma \leftarrow \text{Sign}(\text{SK}[i][0], m, \text{ADM})$
 $c \leftarrow \text{Encrypt}(\text{EK}[i][1], m, \sigma)$
 $c'_0 \leftarrow \text{Redact}(c, \text{MOD})$
 $(m', \sigma') \leftarrow \text{Decrypt}(\text{EK}[i][0], c'_0)$
 $\sigma' \leftarrow \text{Sign}(\text{SK}[i][0], m', \text{ADM})$
 $c'_1 \leftarrow \text{Encrypt}(\text{EK}[i][1], m', \sigma')$
return c'_b .

Experiment 4: Transparency Experiment for Ciphertexts of SEEDS Schemes

Definition 7 (Transparency) A SEEDS scheme is transparent, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\left| \Pr \left[\text{Exp}_{\text{SEEDS}, \mathcal{A}}^{\text{Trans}}(1^\kappa) = 1 \right] - 1/2 \right| < \varepsilon(\kappa)$$

6 Modular instantiation

Scheme 1 instantiates our model by building on generic cryptographic mechanisms, most prominently proxy re-encryption and redactable signatures, which can be instan-

Fig. 3 Graphical overview of our instantiation

tiated with various underlying schemes. Figure 3 illustrates algorithms of Scheme 1.

Signing Instead of signing the plain message parts m_i , we generate a redactable signature over their symmetric ciphertexts c_i . To prevent ambiguity of the actually signed content, we commit to the used symmetric key k with a commitment scheme, giving (O, Com) , and incorporate this commitment Com as another part when generating the redactable signature $\hat{\sigma}$. Neither the ciphertexts of the parts, nor the redactable signature over these ciphertexts, nor the (hiding) commitment reveals anything about the plaintext. To verify, we verify the redactable signature over the ciphertext as well as commitment and check if the message parts decrypted with the committed key match the given message.

Selective sharing We use proxy re-encryption to securely share (encrypt and re-encrypt) the commitment's opening information O with the intended receiver. With the decrypted opening information O and the commitment Com itself, the receiver reconstructs the symmetric key k , which can decrypt the ciphertexts into message parts. In between, redaction can be directly performed on the redactable signature over symmetric ciphertexts and the hiding commitment.

Admissible redactions The admissible redactions ADM describe a set of parts that must not be redacted. For the redactable signature scheme, a canonical representation of this set also has to be signed and later verified against the remaining parts. In combination with proxy re-encryption, the information on admissible redactions also must be protected and is, therefore, part-wise encrypted, which not only allows the receiver to verify the message, but also the proxy to verify if the performed redaction is still valid. Of course, hashes of parts that must remain can be used to describe ADM to reduce its size. In the construction, the commitment Com is added to the signature but must not be redacted, so it is internally added to ADM .

As public parameters, fix a proxy re-encryption scheme PRE, a symmetric encryption scheme SYM, a commitment scheme COM with parameters $c_{pp} \leftarrow \text{COM.KeyGen}(1^\kappa)$, and a redactable signature scheme RSS.

$\text{SignKeyGen}(1^\kappa) \rightarrow (\text{ssk}, \text{spk}):$ **return** $\text{RSS.KeyGen}(1^\kappa)$
 $\text{EncKeyGen}(1^\kappa) \rightarrow (\text{esk}, \text{epk}):$ **return** $\text{PRE.KeyGen}(1^\kappa)$

$\text{ReKeyGen}(\text{esk}_A, \text{epk}_B):$
return $\text{PRE.ReKeyGen}(\text{esk}_A, \text{epk}_B)$

$\text{Sign}(\text{ssk}, m, \text{ADM}) \rightarrow \sigma:$
 $k \leftarrow \text{SYM.KeyGen}(1^\kappa)$
 $c \leftarrow \{\text{SYM.Enc}(k, m_i) \mid m_i \in M\}$
 $c_{\text{ADM}} \leftarrow \{c_i \in c \mid m_i \in \text{ADM}\}$
 $(\text{Com}, O) \leftarrow \text{COM.Commit}(k)$
 $(\cdot, \hat{\sigma}) \leftarrow \text{RSS.Sign}(\text{ssk}, \{\text{Com}\} \cup c, \{\text{Com}\} \cup c_{\text{ADM}})$
return $\sigma \leftarrow (O, \text{Com}, c, \hat{\sigma})$

$\text{Verify}(\text{spk}, m, \sigma) \rightarrow \text{valid} \in \{0, 1\}:$

Parse m as $\{m_i\}_{i=1}^n$
Parse σ as $(O, \text{Com}, \{c_i\}_{i=1}^n, \hat{\sigma})$
if the following holds: **return** 1
 $\text{RSS.Verify}(\text{spk}, \{\text{Com}\} \cup c, \hat{\sigma}) = 1$
 $\forall i \in [1..n]: m_i = \text{SYM.Dec}(k, c_i)$
with $k \leftarrow \text{COM.Open}(\text{Com}, O)$
else: return 0

$\text{Encrypt}(\text{epk}_A, m, \sigma) \rightarrow c_A:$

Parse σ as $(O, \text{Com}, c, \hat{\sigma})$
 $O_A \leftarrow \text{PRE.Enc}^2(\text{epk}_A, O)$
return $c_A \leftarrow (O_A, \text{Com}, c, \hat{\sigma})$

$\text{Decrypt}(\text{esk}_A, c_A) \rightarrow (m, \sigma):$

Parse c_A as $(O_A, \text{Com}, \{c_i\}_{i=0}^n, \hat{\sigma})$
 $O \leftarrow \text{PRE.Dec}^j(\text{esk}_A, O_A)$, with j as ciphertext-level
 $k \leftarrow \text{COM.Open}(O, \text{Com})$
 $m \leftarrow \{m_i\}_{i=1}^n, m_i \leftarrow \text{SYM.Dec}(k, c_i)$
 $\sigma \leftarrow (O, \text{Com}, c, \hat{\sigma})$
if $\text{Verify}(\text{spk}, m, \sigma) = 1$: **return** (m, σ)
else: return $\perp$
We assume that spk can always be correctly derived from any ciphertext c_A .

$\text{Redact}(c_A, \text{MOD}) \rightarrow c'_A:$

Parse c_A as $(O_A, \text{Com}, c, \hat{\sigma})$
 $(\{\text{Com}\} \cup c', \hat{\sigma}') \leftarrow \text{RSS.Redact}(\{\text{Com}\} \cup c, \hat{\sigma}, \text{MOD})$
return $c'_A \leftarrow (O_A, \text{Com}, c', \hat{\sigma}')$

$\text{ReEncrypt}(\text{rk}_{A \rightarrow B}, c_A) \rightarrow c_B:$

Parse c_A as $(O_A, \text{Com}, c, \hat{\sigma})$
 $O_B \leftarrow \text{PRE.ReEnc}(\text{rk}_{A \rightarrow B}, O_A)$
return $c_B \leftarrow (O_B, \text{Com}, c, \hat{\sigma})$

Scheme 1: Modular Instantiation

Subject binding The signature is completely uncoupled from the encryption, and so anyone who obtains or decrypts a signature may encrypt it for herself again. Depending on the use case, the signed data may need to be bound to a specific subject, to describe that this data is about that user. To achieve this, the issuer could specify the subject within the document's signed content. One example would be to add the owner's epk as the first message item, enabling receivers to authenticate supposed owners by engaging in a challenge-response protocol over their spk . As we aim to offer a generic

construction, a concrete method of subject binding is left up to the specific application.

Tailoring The modular design enables to instantiate the building blocks with concrete schemes that best fit the envisioned application scenario. To support the required data structures, a suitable RSS may be selected. Also, performance and space characteristics are a driving factor when choosing suitable schemes. For example, in the original RSS from Johnson et al. [27] the signature grows with each part that is redacted starting from a constant size, while in the (not optimized) RSS from Derler et al. [21, Scheme 1], the signature shrinks with each redaction. Further, already deployed technologies or provisioned key material may come into consideration to facilitate migration. This modularity also becomes beneficial when it is desired to replace a cryptographic mechanism with a related but extended concept. For example, when moving from “classical” PRE to conditional PRE [38] that further limits the proxy's power.

Theorem 1 *Scheme 1 is unforgeable, proxy-private, receiver-private, and transparent, if the used PRE is IND-RCCA2 secure, SYM is IND-CPA secure, COM is binding and hiding, and RSS is unforgeable, private, and transparent.*

The proof is given in Appendix B.

7 Performance

We evaluate the practicability of Scheme 1 by developing and benchmarking three implementations that differ in the used RSS and accumulator schemes. To give an impression for multiple scenarios, we test with various numbers of parts and part sizes, ranging from small identity cards with 10 attributes to 100 measurements à 1kB and from documents with 5 parts à 200 kB to 50 high-definition X-ray scans à 10MB.

Implementations Our three implementations of Scheme 1 aim for 128 bit security. Table 1 summarizes the used cryptographic schemes and their parametrization according to recommendations from NIST [4] for factoring-based and symmetric-key primitives. Groups for pairing-based curves are chosen following recent recommendations [3, 31]. These implementations were developed for the Java platform using the IAIK-JCE and ECCelerate libraries.² We selected the accumulators based on the comparison by Derler et al. [17].

Evaluation methodology In each benchmark, we redact half of the parts. While Redact and ReEnc are likely to be executed on powerful computers, for example in the cloud, the other

² <https://jce.iaik.tugraz.at/>.

Table 1 Implementations and their building blocks

Common	PRE	Chow et al. [15]	3072bit
	SYM	AES-CBC	128 bit
	COM	Hash Commitment (SHA3)	256 bit
	Hash	SHA3	256 bit
Impl. 1	RSS	DPSS [21, Scheme 1]	
	Accu.	CL [10]	3072 bit
	DSS	RSA	3072 bit
Impl. 2	RSS	DPSS [21, Scheme 1]	
	Accu.	DHS [18, Scheme 3]	384 bit
	DSS	ECDSA	256 bit
Impl. 3	RSS	DPSS [21, Scheme 2]	
	Accu.	CL [10]	3072 bit
	DSS	RSA	3072 bit

cryptographic operations might be performed by less powerful mobile phones. Therefore, we performed the benchmarks on three platforms: a PC, an Amazon Web Service machine, and an Android mobile phone. Table 2 summarizes the execution times of the different implementations for all three platforms, where we took the average of 10 runs with different randomly generated data. Instead of also performing the signature verification within the Dec algorithm, we list Verify separately. We had to skip the 500 MB test on the phone, as this exceeds the memory usage limit for apps on our phone.

General observations The growth of execution times is caused by two parameters: the number of parts and the size of the individual parts. Sign symmetrically encrypts all parts and hashes the ciphertexts, so that the RSS signature can then be generated independently of the part sizes. Verify not only hashes the ciphertexts of all remaining parts to verify them against the RSS signature but also symmetrically decrypts the ciphertexts to match them to the plain message. Redact shows very different characteristics in the individual implementations. In contrast, the times for Enc and ReEnc respectively are almost identical, independent of both parameters, as they only perform a single PRE operation on the commitment's opening information from which the symmetric key can be reconstructed. Dec again depends on the number and size of (remaining) parts, as besides the single PRE decryption, all remaining parts are symmetrically decrypted.

Impl. 1 for sets using CL accumulators The first implementation provides the best overall performance. We use an RSS scheme for sets [21, Scheme 1] with CL accumulators [10], where we hash the message parts before signing. This accumulator allows to optimize the implementation (c.f. [21]) to generate a batch witness against which multiple values can be verified at once. These batch operations are considerably more efficient than generating and verifying witnesses for each part. However, with this optimization, it becomes necessary to update the batch witness during the Redact operation.

As only a single witness needs to be stored and transmitted, the RSS signature size is constant.

Impl. 2 for sets using DHS accumulators In the second implementation, we use the same RSS scheme for sets [21, Scheme 1] but move towards elliptic curves by instantiating it with ECDSA signatures and DHS accumulators [18, Scheme 3] (extended version of [17]), which is a variant of Nguyen's accumulator [32]. This accumulator does not allow for the optimization used in the first implementation. Consequently, Redact is very fast, as no witnesses need to be updated. Instead, a witness has to be generated and verified per part. On the PC and server, Sign is slightly faster compared to the first implementation, as signing with ECDSA, evaluating a DHS accumulator, and creating even multiple witnesses is overall more efficient. However, the witness verification within Verify is more costly, which causes a significant impact with a growing number of parts. Interestingly, our phone struggles with the implementation of this accumulator, resulting in far worse times than the otherwise observed slowdown compared with the PC and server. Considering space characteristics, while it is necessary to store one witness per part instead of a single batch witness, each DHS witness is only a single EC point which requires significantly less space than a witness from the CL scheme. Assuming 384-bit EC (compressed) points per witness and an EC point for the DHS accumulator, compared to one 3072-bit CL accumulator and batch witness, the break-even point lies at 15 parts.

Impl. 3 for lists using CL accumulators For the third implementation, we focused on supporting ordered data by using an RSS scheme for lists [21, Scheme 2], while otherwise the same primitives as in our first implementation are used. Of course, with a scheme for sets, it would be possible to encode the ordering for example by appending an index to the parts. However, after redaction, a gap would be observable, which breaks transparency. Achieving transparency for ordered data comes at a cost: Scheme 2 of Derler et al. [21] requires additional accumulators and witnesses updates to keep track of the ordering without breaking transparency, which of course leads to higher computation and space requirements compared to the first implementation. Using CL accumulators also allows for an optimization [21] around batch witnesses and verifications, which reduces the RSS signature size from $O(n^2)$ to $O(n)$.

8 Application of SEEDS

This section describes example use cases where SEEDS could be applied and then elaborates how ordered and hierarchical data structures can be used encoded in a compatible format.

Table 2 Execution times (in milliseconds) of three implementations for Scheme 1 (Dec* denotes decryption without additional signature verification)

#	Size	Impl. 1: Sets & CL					Impl. 2: Sets & DHS					Impl. 3: Lists & CL							
		Sign	Enc	Redact	ReEnc	Dec*	Verify	Sign	Enc	Redact	ReEnc	Dec*	Verify	Sign	Enc	Redact	ReEnc	Dec*	Verify
PC (Intel i7-4790, 3.6 GHz, 16GB RAM)																			
10 ×	1kB	30	1	15	1	1	17	11	1	< 1	1	< 1	49	38	1	34	1	1	19
25 ×	1 kB	54	1	28	1	1	28	20	1	< 1	1	< 1	92	99	1	130	1	1	42
100 ×	1 kB	178	1	95	1	1	89	65	2	< 1	1	1	309	829	1	1582	1	1	261
5 ×	200 kB	22	1	6	1	3	15	13	1	< 1	1	3	42	27	1	12	1	3	17
25 ×	1 MB	133	1	39	1	39	76	103	1	< 1	1	33	148	185	1	148	1	40	98
50 ×	10 MB	1881	1	402	1	744	1050	1771	2	< 1	1	735	1160	2206	1	874	1	638	1182
Amazon web service (c5.4 × large)																			
10 ×	1 kB	26	1	15	1	< 1	13	8	1	< 1	1	< 1	41	34	1	31	1	< 1	17
25 ×	1 kB	44	1	24	1	1	23	12	1	< 1	1	< 1	47	68	1	127	1	< 1	36
100 ×	1 kB	141	1	85	1	1	71	35	1	< 1	1	1	153	444	1	1570	1	1	159
5 ×	200 kB	18	1	7	1	2	10	8	1	< 1	1	2	28	21	1	13	1	2	13
25x	1MB	88	1	38	1	15	52	60	1	< 1	1	15	76	110	1	143	1	16	64
50 ×	10 MB	866	1	308	1	282	627	790	1	< 1	1	298	664	1026	1	753	1	271	665
Mobile phone (OnePlus 6T)																			
10 ×	1 kB	167	9	111	5	15	103	581	11	< 1	6	15	1725	189	9	176	5	15	116
25 ×	1 kB	336	10	180	5	17	191	949	9	< 1	5	15	3680	433	9	551	5	16	229
100 ×	1 kB	1164	9	689	5	23	593	2861	10	1	6	21	13351	2551	9	6091	5	26	936
5 ×	200 kB	133	10	58	5	18	95	463	10	< 1	5	17	1513	142	9	88	5	18	101
25 ×	1 MB	630	9	266	5	79	381	1236	10	< 1	6	74	3932	755	9	668	5	79	430

Fig. 4 Example use cases for SEEDS

8.1 Example use cases

The following use cases illustrate use cases for SEEDS within the banking context (c.f. Fig. 4). We assume that the user (as data owner) has set up an account with a cloud storage service. Various data providers (acting as issuers and senders) upload encrypted and signed data to the user's cloud account, from where subsets of these data can be shared with receivers as defined by the user.

① *Customer identity* Initially, a trusted identity provider certifies identity attributes about the user. This identity provider creates an identity document containing these attributes in individual parts, signs the document (acting as issuer), encrypts the document, and uploads it to the user's cloud account (as sender). The identity provider could be a government agency issuing an electronic version of the user's passport or driver's license.

When the user wants to set up an account with a bank, the bank requires identity information about the user. The user can instruct her cloud storage (acting as proxy) to forward the relevant information by redacting and re-encrypting her identity document. With this document, the bank is able to decrypt the remaining parts and verify their authenticity (acting as receiver). Of course, an additional mechanism has to be employed to ensure that the submitting user is also the owner and subject of the identity document, e.g. a challenge-response protocol on a public key that is included in the identity document.

② *Expenses* The bank could also act as issuer and sender by generating a typical bank account statement for the user as shown in Fig. 5. After signing the statement's parts, the bank uploads the encrypted document to the user's cloud account, where she is able to check her recent transactions.

```
{ "owner": "John Doe",
  "date": "2020-02-20",
  "balance": 25700,
  "txn": [
    { "date": "2020-02-16",
      "name": "taxi",
      "amount": 22 },
    { "date": "2020-02-14",
      "name": "hospital",
      "amount": 230 },
    { "date": "2020-02-13",
      "name": "flight",
      "amount": 425 },
    ... ] }
```

Fig. 5 Bank statement

Such bank statements can be used to get a refund for her expenses on a business trip. The user instructs her cloud account to redact the private parts of the statement (e.g., private payments, account details) and to only keep the relevant transactions as expenses. The cloud storage re-encrypts the remaining parts for the employer, who is able to decrypt and verify the remaining parts, and efficiently refund the user's expenses.

③ *Proof of funds* The illustrated bank statement (c.f. Fig. 5) also contains additional details about the user's bank account, such as the account owner's name as well as the total balance. These signed data items could be used to prove sufficient funds within, e.g., a VISA application process. The individual transactions are redacted to protect the user's privacy.

8.2 Encoding of ordered and hierarchical data

An encoding layer enables to transform hierarchical and ordered data formats, e.g. JSON and XML, into a part-based format that can be used with SEEDS.

```

m = {((owner),      John Doe)
      ((date),      2020-02-20)
      ((balance),   25700)
      ((txn, r1, date), 2020-02-16)
      ((txn, r1, name), taxi)
      ((txn, r1, amount), 22)
      ((txn, r2, date), 2020-02-14)
      ((txn, r2, name), hospital)
      ((txn, r2, amount), 230)
      ((txn, r3, date), 2020-02-13)
      ((txn, r3, name), flight)
      ((txn, r3, amount), 425)}

```

Fig. 6 Hierarchy as parts

```

m = {(owner),      John Doe)
      (date),      2020-02-20)
      (balance),   25700)
      (txn, r1),   {"date": "2020-02-16",
                    "name": "taxi",
                    "amount": 22} )
      (txn, r2),   {"date": "2020-02-14",
                    "name": "hospital",
                    "amount": 230} )
      (txn, r3),   {"date": "2020-02-13",
                    "name": "flight",
                    "amount": 425} )}

```

Fig. 7 Customized granularity

Ordered data Naively, by encoding values together with their indices, the correct sequence of values can be reconstructed and verified even if some parts were redacted in between. However, this approach breaks the transparency property, as the receiver learns that parts were redacted from missing indices. Instead, we instantiate Scheme 1 with an RSS for lists (e.g., Impl. 3) and insert the parts in the correct order. From such an instance for lists, it is not possible to deduce if or where a part was redacted in a received message, but the receiver is still able to verify that the received parts are in the correct order.

Hierarchical data To support hierarchical data structures, we encode the hierarchy into a list of message parts. Figure 6 illustrates the encoding of the example banking statement (c.f. Fig. 5). The encoding layer transforms the hierarchy's items into tuples of: (1) the item's position in the hierarchy as a list of keywords from the root to the item, and (2) the item's value. For items within a list, we additionally randomly choose r_i and add it to the list of keywords. From a collection of such tuples, individual tuples can be redacted to remove subtrees and array items, while it is still possible to reconstruct the remaining partial trees and correctly ordered partial arrays. If the hierarchical data does not con-

tain ordered data, it is sufficient to use an underlying RSS for sets instead of the RSS for lists.

8.3 Encoding with customizable granularity

This section discusses how to customize the granularity of redactable parts in a hierarchy to the needs of an application. The previously described mapping for hierarchical data allows fine-granular redaction on every level of the hierarchy. By customizing the granularity to the needs of the application, we aim to reduce the number of translated message parts, which results in lower storage and communication costs, as well as decreased computation efforts.

An encoding layer separates the data's hierarchy from the redactable parts represented by tuples, as shown in Fig. 7. Instead of encoding each individual node as a tuple, we encode whole subtrees as tuples of the form: (1) the subtree's position in the hierarchy as a list of keywords from the root of the whole tree to the root of the subtree, and (2) the subtree in its original format. Based on the list of keywords, the subtrees can be inserted into the overall hierarchy. It is also possible to use specify further redactable values as children of such subtrees. In conclusion, what parts can be redacted is defined independently of the hierarchy of the data, which enables to perform cryptographic operations on whole blocks containing subtrees rather than on each individual value, and, therefore, increase performance and reduce storage costs.

9 Conclusion

In this paper, we introduced selective end-to-end data-sharing, which covers various issues for data-sharing in honest-but-curious cloud environments by providing end-to-end confidentiality, authenticity, and selective disclosure. First, we formally defined the concept and modeled requirements for cloud data-sharing as security properties. We then instantiated this model with a proven-secure modular construction that is built on generic cryptographic mechanisms, which can be instantiated with various schemes allowing for implementations tailored to the needs of different application domains. We evaluated the performance characteristics of three implementations to highlight the practical usefulness of our modular construction and model as a whole. Finally, we illustrated the application of our cryptographic mechanism for three example use cases and proposed a suitable encoding for hierarchical data structures.

Acknowledgements Open access funding provided by Graz University of Technology. This work was supported by the H2020 EU project CREDENTIAL under Grant agreement number 653454.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adap-

tation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

A Cryptography background

A.1 Proxy re-encryption

This section recalls the security properties of proxy re-encryption to supplement the previously given description of algorithms.

Correctness For correctness, we require that every ciphertext can be decrypted with the respective secret, and, when also considering re-encryptable and re-encrypted ciphertexts, we require that level-2 ciphertexts can be re-encrypted with a suitable re-encryption key and then decrypted using the (delegatee's) respective secret key. More formally, we have, for all security parameters $\kappa \in \mathbb{N}$, all public parameters $\text{pp} \leftarrow \text{Setup}(1^\kappa)$, any number of users $U \in \mathbb{N}$, all key tuples $(\text{pk}_u, \text{sk}_u)_{u \in [U]}$ generated by $\text{KeyGen}(1^\kappa)$, for any $u \in [U]$, all $u' \in [U]$, $u \neq u'$, any re-encryption $\text{rk}_{u \rightarrow u'}$ generated using ReKeyGen from secret keys as well as the target public key, and all messages $m \in \mathcal{M}$, it holds that

$$\forall j \in [2] \exists j' \in [2] : \text{Dec}^{j'}(\text{sk}_u, \text{Enc}^j(\text{pk}_u, m)) = m, \\ \text{Dec}^1(\text{sk}_{u'}, \text{ReEnc}(\text{rk}_{u \rightarrow u'}, \text{Enc}^2(\text{pk}_u, m))) = m.$$

Oracles The experiments use the oracle defined below. The environment maintains initially empty lists of challenge (CH), corrupt (CU) and honest users (HU), as well as initially empty sets EK for encryption keys.

$\text{AHU}(i)$:

if $i \in \text{HU} \cup \text{CU} \cup \text{CH}$: **return** $\perp$
 $\text{HU} \leftarrow \text{HU} \cup \{i\}$, $\text{EK}[i] \leftarrow (\text{esk}_i, \text{epk}_i) \leftarrow \text{KeyGen}(1^\kappa)$
return epk_i

$\text{ACU}(i)$:

if $i \in \text{HU} \cup \text{CU} \cup \text{CH}$: **return** $\perp$
 $\text{CU} \leftarrow \text{CU} \cup \{i\}$, $\text{EK}[i] \leftarrow (\text{esk}_i, \text{epk}_i) \leftarrow \text{KeyGen}(1^\kappa)$
return $(\text{esk}_i, \text{epk}_i)$

$\text{RKG}(i, j)$:

if $i \in \text{CH}$ and $j \in \text{CU}$: **return** $\perp$
if $i, j \notin \text{HU} \cup \text{CU} \cup \text{CH}$ or $i = j$: **return** $\perp$
return $\text{ReKeyGen}(\text{EK}[i][0], \text{EK}[j][1])$.

$\text{RE}(i, j, c_i)$:

if $i, j \notin \text{HU} \cup \text{CU} \cup \text{CH}$ or $i = j$: **return** $\perp$
if c_i is not a proper 2^{nd} level ciphertext for $\text{EK}[i][0]$:
return $\perp$
if the oracle is called in the guess phase, and c_i is a derivative of C^* , that is if $\text{Dec}^2(\text{EK}[i][0], c_i) \in \{m_0, m_1\}$:
return $\perp$
return $\text{ReEnc}(\text{ReKeyGen}(\text{EK}[i][0], \text{EK}[j][1]), c_i)$.

$\text{D}(i, c_i)$:

if $i \notin \text{HU} \cup \text{CU} \cup \text{CH}$: **return** $\perp$
if c_i is not a proper j^{th} level ciphertext for $\text{EK}[i][0]$:
return $\perp$
if the oracle is called in the guess phase, and c_i is a derivative of C^* , that is if $\text{Dec}^j(\text{EK}[i][0], c_i) \in \{m_0, m_1\}$:
return $\perp$
return $m \leftarrow \text{Dec}^j(\text{EK}[i][0], c_i)$

Experiment $\text{Exp}_{\text{PRE}, \mathcal{A}}^{\text{IND-RCCA-2}}(1^\kappa)$

$\text{pp} \leftarrow \text{Setup}(1^\kappa)$, $\text{CH} \leftarrow \{0\}$
 $\text{EK}[0] \leftarrow (\text{sk}^*, \text{pk}^*) \leftarrow \text{KeyGen}(\text{pp})$
 $\mathcal{O}_1 \leftarrow \{\text{AHU}, \text{ACU}, \text{RKG}, \text{RE}, \text{D}\}$, $\mathcal{O}_2 \leftarrow \{\text{RE}, \text{D}\}$
 $(m_0, m_1, \text{st}) \leftarrow \mathcal{A}^{\mathcal{O}_1}(\text{pp}, \text{pk}^*)$
if $m_0 \approx m_1$: **abort**
 $b \xleftarrow{R} \{0, 1\}$, $c^* = \text{Enc}^2(\text{pk}^*, m_b)$
 $b' \leftarrow \mathcal{A}^{\mathcal{O}_2}(\text{st}, c^*)$
if $b = b'$: **return** 1, **else**: **return** 0

Experiment 5: The IND-RCCA security experiment for level 2 ciphertexts of PRE schemes [15].

Definition 8 (IND-RCCA-2) A PRE scheme is IND-RCCA-2 secure if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\left| \Pr \left[\text{Exp}_{\text{PRE}, \mathcal{A}}^{\text{IND-RCCA-2}}(1^\kappa) = 1 \right] - 1/2 \right| < \varepsilon(\kappa).$$

A.2 Redactable signatures

We recall the correctness and security definitions of redactable signature schemes following the notion of [21].

Correctness A redactable signature scheme is correct, if for all $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\kappa)$, all $m \in \mathcal{M}$, for all ADM, and for all $(\sigma, \text{RED}) \leftarrow \text{Sign}(\text{sk}, m, \text{ADM})$, we have that $\text{Verify}(\text{pk}, m, \sigma) = 1$. Additionally, for all allowed modifications MOD, and all $(m^*, \sigma^*, \text{RED}') \leftarrow \text{Redact}(\text{pk}, m, \sigma, \text{MOD}, \text{RED})$, we have that $\text{Verify}(\text{pk}, m^*, \sigma^*) = 1$.

Experiment $\text{Exp}_{\text{RSS}, \mathcal{A}}^{\text{Unf}}(1^\kappa)$
 $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\kappa)$
 $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\text{Sign}(\text{sk}, \cdot, \cdot)}(\text{pk})$ where the environment keeps track of all queries to Sign in Sigs
if $\exists(m, \text{ADM}) \in \text{Sigs}, \exists \text{MOD} \preceq m : m^* \xrightarrow{\text{MOD}} m$:
 return 0
if $\text{Verify}(\text{pk}, m^*, \sigma^*) = 1$: **return** 1, **else**: **return** 0

Experiment 6: The unforgeability experiment of RSS schemes.

Definition 9 (Unforgeability) A RSS scheme is unforgeable if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr \left[\text{Exp}_{\text{RSS}, \mathcal{A}}^{\text{Unf}}(1^\kappa) = 1 \right] < \varepsilon(\kappa).$$

Experiment $\text{Exp}_{\text{RSS}, \mathcal{A}}^{\text{Priv}}(1^\kappa)$
 $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\kappa), b \xleftarrow{R} \{0, 1\}$
 $\mathcal{O} \leftarrow \{\text{Sign}(\text{sk}, \cdot, \cdot), \text{LoRRedact}(\text{sk}, \text{pk}, \cdot, \cdot, b)\}$
 $b^* \leftarrow \mathcal{A}^{\mathcal{O}}(\text{pk})$
if $b^* = b$: **return** 1, **else**: **return** 0
 $\text{LoRRedact}(\text{sk}, \text{pk}, m_0, \text{ADM}_0, \text{MOD}_0, m_1, \text{ADM}_1, \text{MOD}_1, b)$:
 for $c \in \{0, 1\}$
 $(\sigma_c, \text{RED}_c) \leftarrow \text{Sign}(\text{sk}, m_c, \text{ADM}_c)$
 $(m'_c, \sigma'_c, \text{RED}'_c) \leftarrow \text{Redact}(\text{pk}, \sigma_c, m_c, \text{MOD}_c, \text{RED}_c)$
 if $m'_0 \neq m'_1$ or $\text{ADM}'_0 \neq \text{ADM}'_1$: **return** $\perp$
 return (m'_b, σ'_b)

Experiment 7: The privacy experiment of RSS schemes.

Definition 10 (Privacy) A PRE scheme is private if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\left| \Pr \left[\text{Exp}_{\text{RSS}, \mathcal{A}}^{\text{Priv}}(1^\kappa) = 1 \right] - 1/2 \right| < \varepsilon(\kappa).$$

Definition 11 (Transparency) A PRE scheme is transparent if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\left| \Pr \left[\text{Exp}_{\text{RSS}, \mathcal{A}}^{\text{Trans}}(1^\kappa) = 1 \right] - 1/2 \right| < \varepsilon(\kappa).$$

Experiment $\text{Exp}_{\text{RSS}, \mathcal{A}}^{\text{Trans}}(1^\kappa)$
 $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\kappa), b \xleftarrow{R} \{0, 1\}$
 $\mathcal{O} \leftarrow \{\text{Sign}(\text{sk}, \cdot, \cdot), \text{RedactOrNot}(\text{sk}, \text{pk}, \cdot, \cdot, b)\}$
 $b^* \leftarrow \mathcal{A}^{\mathcal{O}}(\text{pk})$
if $b^* = b$: **return** 1, **else**: **return** 0
 $\text{RedactOrNot}(\text{sk}, \text{pk}, m, \text{ADM}, \text{MOD}, b)$:
 $(\sigma, \text{RED}) \leftarrow \text{Sign}(\text{sk}, m, \text{ADM})$
 $(m', \sigma_0, \text{RED}') \leftarrow \text{Redact}(\text{pk}, \sigma, m, \text{MOD}, \text{RED})$
 $(\sigma_1, \text{RED}'') \leftarrow \text{Sign}(\text{sk}, m', \text{ADM}')$
 return (m'_b, σ'_b)

Experiment 8: The transparency experiment of RSS schemes.

A.3 Commitments

We briefly recall the definition and security notions of commitment schemes.

Definition 12 (Commitment scheme) A (non-interactive) commitment scheme for message space $\mathcal{M}$ consists of three algorithms KeyGen , Commit , Open with the following properties:

$\text{KeyGen}(1^\kappa) \rightarrow \text{pk}$: On input the security parameter κ , outputs a public key pk .³

$\text{Commit}(m) \rightarrow (\text{Com}, \mathcal{O})$: On input of a message $m \in \mathcal{M}$, computes a commitment Com and opening $\mathcal{O}$, and outputs $(\text{Com}, \mathcal{O})$.

$\text{Open}(\text{Com}, \mathcal{O}) \rightarrow m$: On input of commitment Com and opening information $\mathcal{O}$, the opening algorithm either outputs a message $m \in \mathcal{M}$ or $\perp$.

Besides the obvious correctness of an commitment scheme, we require commitments to be (computationally) binding and hiding:

Experiment $\text{Exp}_{\text{COM}, \mathcal{A}}^{\text{Hide}}(1^\kappa)$
 $\text{pk} \leftarrow \text{Setup}(1^\kappa), (m_0, m_1, \text{st}) \leftarrow \mathcal{A}(\text{pk})$,
 $b \xleftarrow{R} \{0, 1\}, (\mathcal{O}, \text{Com}) \leftarrow \text{Commit}(m_b)$
 $b^* \leftarrow \mathcal{A}(\text{Com}, \text{st})$
if $b^* = b$: **return** 1, **else**: **return** 0

Experiment 9: The hiding security experiment of COM schemes.

Definition 13 (Hiding) A COM scheme is hiding if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\left| \Pr \left[\text{Exp}_{\text{COM}, \mathcal{A}}^{\text{Hide}}(1^\kappa) = 1 \right] - \frac{1}{2} \right| \leq \varepsilon(\kappa).$$

Definition 14 (Binding) A COM scheme is binding if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr \left[\text{Exp}_{\text{COM}, \mathcal{A}}^{\text{Bind}}(1^\kappa) = 1 \right] \leq \varepsilon(\kappa).$$

Experiment $\text{Exp}_{\text{COM}, \mathcal{A}}^{\text{Bind}}(1^\kappa)$
 $\text{pk} \leftarrow \text{Setup}(1^\kappa), (\text{Com}, \mathcal{O}_0, \mathcal{O}_1) \leftarrow \mathcal{A}(\text{pk})$
 $m_0 \leftarrow \text{Open}(\mathcal{O}_0, \text{Com}), m_1 \leftarrow \text{Open}(\mathcal{O}_1, \text{Com})$
if $m_0, m_1 \neq \perp$ and $m_0 \neq m_1$: **return** 1
else: **return** 0

Experiment 10: The binding experiment of COM schemes.

³ We assume pk to be an implicit input to Commit and Open .

B Proof of Theorem 1

We prove Theorem 1 by proving Lemmas 1–4 to show the properties unforgeability, proxy privacy, receiver privacy, and, finally, transparency. For proofs using a sequence of games, we denote the event that an adversary wins game i by S_i .

Lemma 1 *Scheme 1 is unforgeable if RSS is unforgeable and COM is binding.*

Proof We prove this lemma using a sequence of games.

Game 0: The original SEEDS unforgeability game.

Game 1: We adapt Game 0 to also abort when the signatures were generated by SIG.

SIG(i, m, ADM):

```

if  $i \notin \text{CH} \cup \text{HU}$  or  $\text{SK}[i][0] = \perp$ : return  $\perp$ 
 $\sigma \leftarrow \text{Sign}(\text{SK}[i][0], m, \text{ADM})$ 
Parse  $\sigma$  as  $(O, \text{Com}, c, \hat{\sigma})$ 
 $\text{Sigs} \leftarrow \text{Sigs} \cup \{m' \mid \forall \text{MOD} \stackrel{\text{ADM}}{\preceq} m \forall m' \stackrel{\text{MOD}}{\longleftarrow} m\}$ 
 $\text{Coms} \leftarrow \text{Coms} \cup \{\text{Com}\}$ 
return  $(m, \sigma)$ 

```

Game 1:

```

 $(\text{ssk}^*, \text{spk}^*) \leftarrow \text{SignKeyGen}(1^\kappa)$ 
 $\text{CH} \leftarrow \{0\}, \text{SK}[0] \leftarrow (\text{ssk}^*, \text{spk}^*)$ 
 $\mathcal{O} \leftarrow \{\text{SIG}\}$ 
 $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(\text{spk}^*)$ 
Parse  $\sigma$  as  $(O, \text{Com}, c, \hat{\sigma})$ 
if  $\text{Com} \in \text{Coms}$ : return 0
if  $\text{Verify}(\text{spk}^*, m, \sigma) = 1$  and  $m \notin \text{Sigs}$ : return 1
else: return 0

```

Transition $0 \Rightarrow 1$: Game 1 behaves the same as Game 0 unless $\mathcal{A}$ returned a valid pair (m, σ) where the included RSS signature $\hat{\sigma}$ on $\{\text{Com}\} \cup c$ was generated by SIG. We denote this failure event as F , thus $|\Pr[S_0] - \Pr[S_1]| \leq \Pr[F]$. In this case, since Com , c , and $\hat{\sigma}$ are fixed, two different messages can only be obtained, by decrypting with different keys k_1 and k_2 . From the fixed Com , different keys can only be recovered with different openings O_1 and O_2 . To achieve this, the adversary would have to break the binding property of COM, therefore $\Pr[F] = \epsilon_{\text{COM}}^{\text{Bind}}(\kappa)$.

Finally, we build an efficient adversary $\mathcal{B}$ from an adversary $\mathcal{A}$ winning Game 1 for the unforgeability of RSS in $\mathcal{R}_{\text{RSS}}^{\text{Unf}} \rightarrow G_1$. We can simulate SIG except for $i = 0$, where we obtain the RSS signatures using its signing oracle. Note that all values are consistently distributed. Now, if we obtain a forgery (m, σ) from $\mathcal{A}$, then parse σ as $(O, \text{Com}, c, \hat{\sigma})$ and forward $\{\text{Com}\} \cup c, \hat{\sigma}$ as a forgery. Therefore, $\Pr[S_1] = \epsilon_{\text{RSS}}^{\text{Unf}}(\kappa)$,

resulting in $\Pr[S_0] = \epsilon_{\text{COM}}^{\text{Bind}}(\kappa) + \epsilon_{\text{RSS}}^{\text{Unf}}(\kappa)$, which is negligible.

Reduction $\mathcal{R}_{\text{RSS}}^{\text{Unf}} \rightarrow G_1(\text{pk})$:

```

 $(m, \sigma) \leftarrow \mathcal{A}^{\text{SIG}}(\text{pk})$ 
Parse  $\sigma$  as  $(O, \text{Com}, c, \hat{\sigma})$ 
return  $(\{\text{Com}\} \cup c, \hat{\sigma})$ 

```

SIG($0, m, \text{ADM}$):

```

 $k \leftarrow \text{SYM.KeyGen}(1^\kappa)$ 
 $c \leftarrow \{\text{SYM.Enc}(k, m_i) \mid m_i \in M\}$ 
 $c_{\text{ADM}} \leftarrow \{c_i \mid c_i \in c, m_i \in \text{ADM}\}$ 
 $\sigma \leftarrow (O, \text{Com}, c, \mathcal{O}^{\text{Sign}}(\text{sk}, \{\text{Com}\} \cup c, \{\text{Com}\} \cup c_{\text{ADM}}))$ 
 $\text{Sigs} \leftarrow \text{Sigs} \cup \{m' \mid \forall \text{MOD} \stackrel{\text{ADM}}{\preceq} m \forall m' \stackrel{\text{MOD}}{\longleftarrow} m\}$ 
 $\text{Coms} \leftarrow \text{Coms} \cup \{\text{Com}\}$ 
return  $(m', \sigma)$ 

```

□

Lemma 2 *If the PRE is IND-RCCA-2 secure, COM is hiding and binding, SYM is IND-CPA secure, and RSS is unforgeable, then Scheme 1 is proxy private.*

Proof We prove proxy privacy using a sequence of games.

Game 0: The original SEEDS proxy privacy game.

Game 1: We restrict the decryption oracles to ciphertexts that contain messages signed by the signature oracles. Therefore, we adapt SE as SIG in Lemma 1 to track the generated commitments, $\text{Coms} \leftarrow \text{Coms} \cup \{\text{Com}\}$.

Also, we adapt D to parse c_i as $(O_A, \text{Com}, c, \hat{\sigma})$ and abort in case **if** $\text{Com} \notin \text{Coms}$: **return** $\perp$.

Transition $0 \Rightarrow 1$: The two games proceed identically unless the adversary submits a valid signature to D. In that case the adversary produced a forgery, i.e. $|\Pr[S_0] - \Pr[S_1]| \leq \epsilon_{\text{SEEDS}}^{\text{Unf}}(\kappa)$.

Game 2: In the used Encrypt algorithm, we replace the opening information with a random r from the same domain, and simulate the oracles accordingly:

Encrypt:

```

Parse  $\sigma$  as  $(O, \text{Com}, c, \hat{\sigma})$ 
 $r \stackrel{R}{\leftarrow} \text{Domain}(\text{COM}_O)$ 
 $O_A \leftarrow \text{PRE.Enc}^2(\text{epk}_A, [r])$ 
 $\text{Map} \leftarrow \text{Map} \cup \{(O_A, \text{Com}, O)\}$ 
return  $c_A \leftarrow (O_A, \text{Com}, c, \hat{\sigma})$ 

```

RE(i, j, k, c_j):

```

Parse  $c_j$  as  $(O_A, \text{Com}, c, \hat{\sigma})$ 

```

Look up $(O_A, Com, O) \in \text{Map}$
if not contained: run RE unmodified
else:
 $O_B \leftarrow \text{PRE.ReEnc}(\text{rk}_{i \rightarrow j}, O_A)$
 $\text{Map} \leftarrow \text{Map} \cup \{(O_B, Com, O)\}$
return $(O_B, Com, c, \hat{\sigma})$

$D(i, c_i)$:

Parse c_i as $(O_A, Com, c, \hat{\sigma})$
 Lookup $(O_A, Com, O) \in \text{Map}$
if not contained: run D unmodified.
else:
 $k \leftarrow \text{COM.Open}(O, Com)$
 $m \leftarrow \{\text{SYM.Dec}(k, c_i) \mid c_i \in c\}$
 $\sigma \leftarrow (O, Com, c, \hat{\sigma})$
if $\text{Verify}(\text{spk}, m, \sigma) \neq 1$ or m is a subset of forwarded m_0, m_1 : **return** $\perp$
return (m, σ)

Transition 1 $\Rightarrow$ 2: From a distinguisher $\mathcal{D}^{1 \rightarrow 2}$, we build an IND-RCCA-2 adversary against the PRE scheme. Indeed, let $\mathcal{C}$ be an IND-RCCA-2 challenger. We modify Encrypt in the following way: Simulate everything honestly, but sample $\boxed{r}$ uniformly at random from the domain of openings of COM and run $\boxed{O_A \leftarrow \mathcal{C}(O, r)}$, where $c \leftarrow \mathcal{C}(m_0, m_1)$ denotes a challenge ciphertext with respect to m_0 and m_1 . The RE oracle calls the challenger's RE oracle instead of PRE.ReEnc. Consequently, we arrive at $|\Pr[S_1] - \Pr[S_2]| \leq \epsilon_{\text{PRE}}^{\text{IND-RCCA-2}}(\kappa)$.

Game 3: For the signature contained in the challenge ciphertext, we commit to a random value. That is, we choose $\boxed{r \xleftarrow{R} \text{Domain}(\text{SYM}_k)}$ and let $(Com, O) \leftarrow \text{COM.Commit}(\boxed{r})$.

Transition 2 $\Rightarrow$ 3: From a distinguisher $\mathcal{D}^{2 \rightarrow 3}$, we obtain a hiding adversary against COM. Let $\mathcal{C}$ be a hiding challenger. We modify Sign in the following way: Simulate everything honestly, but choose $\boxed{r}$ uniformly at random from the same domain as the SYM keys and run $\boxed{Com \leftarrow \mathcal{C}(k, r)}$, where $Com \leftarrow \mathcal{C}(m_0, m_1)$ denotes a challenge commitment with respect to m_0 and m_1 . Therefore, $|\Pr[S_2] - \Pr[S_3]| \leq \epsilon_{\text{COM}}^{\text{Hide}}(\kappa)$.

Game 4: In the challenge ciphertext, we replace the message parts with random values drawn from an identical domain with the same corresponding lengths, i.e. $\boxed{\text{for } i \in [1..|M|]: \quad r_i \xleftarrow{R} \text{Domain}(m)}$ and $c \leftarrow \{\text{SYM.Enc}(k, \boxed{r_i}) \mid m_i \in m\}$.

Transition 3 $\Rightarrow$ 4: A distinguisher $\mathcal{D}^{3 \rightarrow 4}$ is a (hybrid) IND-CPA adversary against SYM. Let $\mathcal{C}$ be an IND-CPA challenger. We modify Sign in the following way: Simulate everything honestly, but for each message part choose $\boxed{r_i}$ uniformly at random from the message space and run

$\boxed{c_i \leftarrow \mathcal{C}(m_i, r_i)}$, where $c \leftarrow \mathcal{C}(m_0, m_1)$ denotes a challenge ciphertext with respect to m_0 and m_1 . Therefore, we have $|\Pr[S_4] - \Pr[S_3]| \leq |m| \cdot \epsilon_{\text{SYM}}^{\text{IND-CPA}}(\kappa)$, with $|m|$ polynomial in the security parameter κ .

Finally, we have that $\Pr[S_4] = 1/2$, since the adversary now cannot do better than guessing. Combining the claims, we see that the following is negligible:

$$|\Pr[S_0] - 1/2| \leq \epsilon_{\text{SEEDS}}^{\text{Unf}}(\kappa) + \epsilon_{\text{PRE}}^{\text{IND-RCCA-2}}(\kappa) + \epsilon_{\text{COM}}^{\text{Hide}}(\kappa) + |m| \cdot \epsilon_{\text{SYM}}^{\text{IND-CPA}}(\kappa)$$

□

Lemma 3 If RSS is private, then Scheme 1 is receiver private.

Proof Assuming there is an efficient adversary $\mathcal{A}$ against the receiver privacy of Scheme 1, we build an adversary $\mathcal{B}$ against the privacy of RSS:

Reduction $\mathcal{R}_{\text{RSS}}^{\text{Priv}} \xrightarrow{\text{SEEDS}}^{\text{RP}}(\text{pk})$:
 $(\text{esk}^*, \text{epk}^*) \leftarrow \text{SEEDS.EncKeyGen}(1^\kappa)$
 $\text{CH} \leftarrow \{0\}$
 $\text{EK}[0] \leftarrow (\text{esk}^*, \text{epk}^*), \text{SK}[0] \leftarrow (\perp, \text{pk})$
 $\mathcal{O} \leftarrow \{\text{AU}, \text{SIG}, \text{LoRRedact}(\dots, b)\}$
return $b^* \leftarrow \mathcal{A}^{\mathcal{O}}(\text{pk}, \text{epk}^*)$

$\text{AU}(i, t)$: This oracle is simulated honestly.

$\text{SIG}(i, m, \text{ADM})$: For $i \in \text{HU}$ and $i \in \text{CU}$ everything is computed honestly, while we use $\mathcal{O}^{\text{Sign}}$ for $i \in \text{CH}$ as in Lemma 1.

$\text{LoRRedact}(i, j, m_0, \text{MOD}_0, m_1, \text{MOD}_1, \text{ADM}, b)$: For $i \neq 0$ or $j \notin \text{HU} \cup \text{CU}$ everything is computed honestly, otherwise we run:

$\text{rk} \leftarrow \text{ReKeyGen}(\text{EK}[i][0], \text{EK}[j][1])$
 $k \leftarrow \text{SYM.KeyGen}(1^\kappa; \omega)$
 $(Com, O) \leftarrow \text{COM.Commit}(k)$
 $O_B \leftarrow \text{PRE.ReEnc}(\text{rk}, \text{PRE.Enc}(\text{EK}[i][1], O))$
for both $c \in \{0, 1\}$:
 $c_c \leftarrow \{\text{SYM.Enc}(k, m_i) \mid m_i \in m_c\}$
 $c_{\text{ADM}, c} \leftarrow \{c_i \mid c_i \in c_c, m_i \in \text{ADM}\}$
 $X \leftarrow \mathcal{O}^{\text{LoRRedact}}(\underline{\text{sk}}, \underline{\text{pk}}, (\{Com\} \cup c_c, \{Com\} \cup c_{\text{ADM}, c}, \text{MOD}_c)_{c \in \{0, 1\}}, \underline{b})$
if $X = \perp$: **return** $\perp$
 Parse X as $(\{Com\} \cup c_{b'}, \hat{\sigma}_{b'})$
return $(O_B, Com, c_{b'}, \hat{\sigma}_{b'})$

The reduction extends the RSS public key to a SEEDS public key, and forwards it to $\mathcal{A}$. The oracle LoRRedact sets up everything honestly and obtains signatures from LoRRedact of RSS. All values are distributed consistently, and $\mathcal{B}$ wins the privacy experiment of RSS with the same probability as $\mathcal{A}$ breaks the SEEDS receiver privacy of Scheme 1. □

Lemma 4 *If RSS is transparent, then Scheme 1 is transparent.*

Proof Assuming there is an efficient adversary $\mathcal{A}$ against the transparency of Scheme 1, we construct an adversary $\mathcal{B}$ against the transparency of the RSS:

Reduction $\mathcal{R}_{\text{RSS}}^{\text{Trans}} \rightarrow^{\text{Trans}}_{\text{SEEDS}}(\text{pk}): (\text{esk}^*, \text{epk}^*) \leftarrow \text{SEEDS.EncKeyGen}(1^\kappa)$
 $\text{CH} \leftarrow \{0\}$
 $\text{EK}[0] \leftarrow (\text{esk}^*, \text{epk}^*), \text{SK}[0] \leftarrow (\perp, \text{pk})$
 $\mathcal{O} \leftarrow \{\text{AU}, \text{SIG}, \text{RedactOrNot}\}$
return $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\text{esk}^*, \text{epk}^*, \text{pk})$
 $\text{AU}(i, t):$ This oracle is simulated honestly.
 $\text{SIG}(i, m, \text{ADM}):$ For $i \in \text{HU}$ and $i \in \text{CU}$ everything is computed honestly, while we use $\mathcal{O}^{\text{Sign}}$ for $i \in \text{CH}$ as in Lemma 1.
 $\text{RedactOrNot}(i, m, \text{MOD}, \text{ADM}, b):$ For $i \neq 0$ everything is computed honestly, otherwise we run the following:
 $k \leftarrow \text{SYM.KeyGen}(1^\kappa; \omega)$
 $(\text{Com}, \mathcal{O}) \leftarrow \text{COM.Commit}(k)$
 $\mathcal{O}_A \leftarrow \text{PRE.Enc}(\text{EK}[i][1], \mathcal{O})$
 $c \leftarrow \{\text{SYM.Enc}(k, m_i) \mid m_i \in m\}$
 $c_{\text{ADM}} \leftarrow \{c_i \mid c_i \in c, m_i \in \text{ADM}\}$
 $(\{\text{Com}\} \cup c', \hat{\sigma}') \leftarrow \mathcal{O}^{\text{Sign/Redact}}(\text{sk}, \text{pk}, \{\text{Com}\} \cup c, \text{MOD}, \{\text{Com}\} \cup c_{\text{ADM}}, b)$
return $c'_{A,b} \leftarrow (\mathcal{O}_A, \text{Com}, c', \hat{\sigma}')$

The reduction extends the RSS public key to a SEEDS public key honestly, and forwards it together with the secret encryption key to $\mathcal{A}$. Similarly, RedactOrNot sets up everything honestly and queries the RSS oracle $\mathcal{O}^{\text{Sign/Redact}}$ to obtain the signature. Finally, it outputs a consistent ciphertext, hence, $\mathcal{B}$ wins with the same probability as $\mathcal{A}$. $\square$

References

- Ateniese G, Benson K, Hohenberger S (2009) Key-private proxy re-encryption. In: M. Fischlin (ed.) Proceedings of topics in cryptology-CT-RSA 2009, the cryptographers' Track at the RSA conference, LNCS, vol 5473, 2009, San Francisco, CA, USA, April 20–24. Springer, pp 279–294. https://doi.org/10.1007/978-3-642-00862-7_19
- Ateniese G, Fu K, Green M, Hohenberger S (2006) Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans Inf Syst Secur 9(1):1–30. <https://doi.org/10.1145/1127345.1127346>
- Barbulescu R, Dquesne S (2019) Updating key size estimations for pairings. J Cryptol 32(4):1298–1336. <https://doi.org/10.1007/s00145-018-9280-5>
- Barker E (2016) SP 800-57. Recommendation for Key Management, Part 1: General (Rev 4). Tech. rep., National Institute of Standards & Technology
- Blaze M, Bleumer G, Strauss M (1998) Divertible protocols and atomic proxy cryptography. In: K. Nyberg (ed.) Proceeding advances in cryptology-EUROCRYPT '98, international conference on the theory and application of cryptographic techniques, LNCS, vol 1403, Espoo, Finland, May 31–June 4. Springer, pp. 127–144. <https://doi.org/10.1007/BFb0054122>
- Boneh D, Sahai A, Waters B (2011) Functional encryption: Definitions and challenges. In: Ishai Y (ed) Proceedings of theory of cryptography-8th theory of cryptography conference, TCC 2011, LNCS, Providence, RI, USA, March 28–30, 2011, vol 6597. Springer, pp 253–273. https://doi.org/10.1007/978-3-642-19571-6_16
- Brzuska C, Busch H, Dagdelen Ö, Fischlin M, Franz M, Katzenbeisser S, Manulis M, Onete C, Peter A, Poettering B, Schröder D (2010) Redactable signatures for tree-structured data: definitions and constructions. In: Zhou J, Yung M (eds) Proceedings of Applied cryptography and network security, 8th international conference, vol 6123, LNCS, ACNS 2010, Beijing, China, June 22–25. pp 87–104. https://doi.org/10.1007/978-3-642-13708-2_6
- Camenisch J, Herreweghen EV (2002) Design and implementation of the idemix anonymous credential system. In: Atluri V (ed) Proceedings of the 9th ACM conference on computer and communications security, CCS 2002, Washington, DC, USA, November 18–22. ACM, pp 21–30. <https://doi.org/10.1145/586110.586114>
- Camenisch J, Lysyanskaya A (2001) An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann B (ed) Advances in cryptology-EUROCRYPT 2001, international conference on the theory and application of cryptographic techniques, vol 2045, Innsbruck, Austria, May 6–10. Springer, pp 93–118. https://doi.org/10.1007/3-540-44987-6_7
- Camenisch J, Lysyanskaya A (2002) Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung M (ed) Proceedings advances in cryptology-CRYPTO 2002, 22nd annual international cryptology conference, vol 2442. LNCS, Santa Barbara, California, USA, August 18–22. Springer, pp 61–76. https://doi.org/10.1007/3-540-45708-9_5
- Canetti R, Hohenberger S (2007) Chosen-ciphertext secure proxy re-encryption. In: Ning P, di Vimercati, SDC, Syverson PF (eds) Proceedings of the 2007 ACM conference on computer and communications security, CCS 2007, Alexandria, Virginia, USA, October 28–31. ACM, pp 185–194. <https://doi.org/10.1145/1315245.1315269>
- Chandran N, Chase M, Liu F, Nishimaki R, Xagawa K (2014) Re-encryption, functional re-encryption, and multi-hop re-encryption: a framework for achieving obfuscation-based security and instantiations from lattices. In: Krawczyk H (ed) Proceedings Public-key cryptography—PKC 2014—17th international conference on practice and theory in public-key cryptography, LNCS, vol 8383, Buenos Aires, Argentina, March 26–28. Springer, pp 95–112. https://doi.org/10.1007/978-3-642-54631-0_6
- Chaum D (1981) Untraceable electronic mail, return addresses, and digital pseudonyms. Commun ACM 24(2):84–88. <https://doi.org/10.1145/358549.358563>
- Chaum D (1985) Security without identification: transaction systems to make big brother obsolete. Commun ACM 28(10):1030–1044. <https://doi.org/10.1145/4372.4373>
- Chow SSM, Weng J, Yang Y, Deng RH (2010) Efficient unidirectional proxy re-encryption. In: Bernstein DJ, Lange T (eds) Proceedings, LNCS progress in cryptology-AFRICACRYPT 2010, third international conference on cryptology in Africa, vol 6055, Stellenbosch, South Africa, May 3–6. Springer, pp 316–332. https://doi.org/10.1007/978-3-642-12678-9_19
- Demirel D, Derler D, Hanser C, Pöhls HC, Slamanig D, Traverso G (2015) PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Tech. rep., H2020 PRISMACLOUD
- Derler D, Hanser C, Slamanig D (2015) Revisiting cryptographic accumulators, additional properties and relations to other primitives. In: Nyberg K (ed) Proceedings, LNCS Topics in cryptology-CT-RSA 2015, the cryptographer's track at the RSA conference

- 2015, vol 9048, San Francisco, CA, USA, April 20–24. Springer, pp 127–144. https://doi.org/10.1007/978-3-319-16715-2_7
18. Derler D, Hanser C, Slamanig D (2015) Revisiting cryptographic accumulators, additional properties and relations to other primitives. IACR Cryptology ePrint Archive, vol 87. <http://eprint.iacr.org/2015/087>. Accessed 18 Sept 2019
19. Derler D, Krenn S, Lorünser T, Ramacher S, Slamanig D, Striecks C (2018) Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. In: Abdalla M, Dahab R (eds) Proceedings, Part I, LNCS public-key cryptography-PKC 2018-21st IACR international conference on practice and theory of public-key cryptography, vol 10769, Rio de Janeiro, Brazil, March 25–29. Springer, pp 219–250. https://doi.org/10.1007/978-3-319-76578-5_8
20. Derler D, Krenn S, Slamanig D (2016) Signer-anonymous designated-verifier redactable signatures for cloud-based data sharing. In: Foresti S, Persiano G (eds) Proceedings, LNCS cryptography and network security-15th international conference, vol 10052, CANS 2016, Milan, Italy, November 14–16. pp 211–227. https://doi.org/10.1007/978-3-319-48965-0_13
21. Derler D, Pöhls HC, Samelin K, Slamanig D (2015) A general framework for redactable signatures and new constructions. In: Kwon S, Yun A (eds) Information security and cryptography-ICISC 2015-18th international conference, LNCS, vol 9558, Seoul, South Korea, November 25–27, Revised Selected Papers. Springer, pp 3–19. https://doi.org/10.1007/978-3-319-30840-1_1
22. Derler D, Ramacher S, Slamanig D (2017) Homomorphic proxy re-authenticators and applications to verifiable multi-user data aggregation. In: Kiayias A (ed) Financial cryptography and data security-21st international conference, LNCS, FC 2017, vol 10322, Sliema, Malta, April 3–7, 2017, Revised Selected Papers. Springer, pp 124–142. https://doi.org/10.1007/978-3-319-70972-7_7
23. European Commission (2016) Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union L119/59
24. Goyal V, Pandey O, Sahai A, Waters B (2006) Attribute-based encryption for fine-grained access control of encrypted data. In: Juels A, Wright RN, di Vimercati SDC (eds) Proceedings of the 13th ACM conference on computer and communications security, CCS 2006, Alexandria, VA, USA, October 30–November 3. ACM, pp 89–98. <https://doi.org/10.1145/1180405.1180418>
25. Hörandner F, Krenn S, Migliaiaccia A, Thieme F, Zwartendorfer B (2016) CREDENTIAL: a framework for privacy-preserving cloud-based data sharing. In: 11th international conference on availability, reliability and security, ARES 2016, Salzburg, Austria, August 31–September 2. IEEE Computer Society, pp 742–749. <https://doi.org/10.1109/ARES.2016.79>
26. Hörandner F, Ramacher S, Roth S (2019) Selective end-to-end data-sharing in the cloud. In: Garg D, Kumar NVN, Shyamasundar RK (eds) Proceedings of information systems security-15th international conference, LNCS, vol 11952, ICIS 2019, Hyderabad, India, December 16–20. Springer, pp 175–195. https://doi.org/10.1007/978-3-030-36945-3_10
27. Johnson R, Molnar D, Song DX, Wagner DA (2002) Homomorphic signature schemes. In: Preneel B (ed) Proceedings topics in cryptography-CT-RSA 2002, the cryptographer's Track at the RSA conference, LNCS, vol 2271, San Jose, CA, USA, February 18–22, 2002. Springer, pp 244–262. https://doi.org/10.1007/3-540-45760-7_17
28. Krenn S, Lorünser T, Salzer A, Striecks C (2017) Towards attribute-based credentials in the cloud. In: Capkun S, Chow SSM (eds) Cryptology and network security-16th international conference, LNCS, vol 11261, CANS 2017, Hong Kong, China, November 30–December 2, 2017, Revised Selected Papers. Springer, pp 179–202. https://doi.org/10.1007/978-3-030-02641-7_9
29. Kundu A, Bertino E (2013) Privacy-preserving authentication of trees and graphs. *Int J Inf Sec* 12(6):467–494. <https://doi.org/10.1007/s10207-013-0198-5>
30. Libert B, Vergnaud D (2008) Unidirectional chosen-ciphertext secure proxy re-encryption. In: Cramer R (ed) Proceedings of PKC-PKC 2008, 11th international workshop on practice and theory in public-key cryptography, LNCS, vol 4939, Barcelona, Spain, March 9–12. Springer, pp 360–379. https://doi.org/10.1007/978-3-540-78440-1_21
31. Menezes A, Sarkar P, Singh S (2016) Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In: Phan RC, Yung M (eds) Paradigms in cryptology-Mycrypt 2016. Malicious and exploratory cryptology-second international conference, LNCS, vol 10311, Mycrypt 2016, Kuala Lumpur, Malaysia, December 1–2, 2016, Revised Selected Papers. Springer, pp 83–108. https://doi.org/10.1007/978-3-319-61273-7_5
32. Nguyen L (2005) Accumulators from bilinear pairings and applications. In: Menezes A (ed) Proceedings of topics in cryptography-CT-RSA 2005, the cryptographers' track at the RSA conference, LNCS, vol 3376, 2005, San Francisco, CA, USA, February 14–18. Springer, pp 275–292. https://doi.org/10.1007/978-3-540-30574-3_19
33. Paquin C, Zaverucha G (2013) U-prove cryptographic specification v1.1 (revision 3). Tech. rep., Microsoft
34. Piretti M, Traynor P, McDaniel PD, Waters B (2006) Secure attribute-based systems. In: Juels A, Wright RN, di Vimercati SDC (eds) Proceedings of the 13th ACM conference on computer and communications security, CCS 2006, Alexandria, VA, USA, October 30–November 3. ACM, pp 99–112. <https://doi.org/10.1145/1180405.1180419>
35. Sahai A, Waters B (2005) Fuzzy identity-based encryption. In: Cramer R (ed) Proceedings of advances in cryptology-EUROCRYPT 2005, 24th annual international conference on the theory and applications of cryptographic techniques, LNCS, vol 3494, Aarhus, Denmark, May 22–26. Springer, pp 457–473. https://doi.org/10.1007/11426639_27
36. Samelin K, Pöhls HC, Bilzhaue A, Posegga J, de Meer H (2012) Redactable signatures for independent removal of structure and content. In: Ryan MD, Smyth B, Wang G (eds) Proceedings of information security practice and experience-8th international conference, LNCS, vol 7232, ISPEC 2012, Hangzhou, China, April 9–12. Springer, pp 17–33. https://doi.org/10.1007/978-3-642-29101-2_2
37. Steinfeld R, Bull L, Zheng Y (2001) Content extraction signatures. In: Kim K (ed) Proceedings information security and cryptography-ICISC 2001, 4th international conference, LNCS, vol 2288. Seoul, Korea, December 6–7. Springer, pp 285–304. https://doi.org/10.1007/3-540-45861-1_22
38. Weng J, Deng RH, Ding X, Chu C, Lai J (2009) Conditional proxy re-encryption secure against chosen-ciphertext attack. In: Li W, Susilo W, Tupakula UK, Safavi-Naini R, Varadharajan V (eds) Proceedings of the 2009 ACM symposium on information, computer and communications security, ASIACCS 2009, Sydney, Australia, March 10–12. ACM, pp 322–332. <https://doi.org/10.1145/1533057.1533100>