Selective End-To-End Data-Sharing in the Cloud

Felix Hörandner[1(✉)], Sebastian Ramacher[2,⋆], and Simon Roth[1]

[1] Graz University of Technology
{felix.hoerandner, simon.roth}@iaik.tugraz.at
[2] AIT Austrian Institute of Technology, Vienna, Austria
Sebastian.Ramacher@ait.ac.at

Abstract. Cloud-based services enable easy-to-use data-sharing between multiple parties, and, therefore, have been widely adopted over the last decade. Storage services by large cloud providers such as Dropbox or Google Drive as well as federated solutions such as Nextcloud have amassed millions of users. Nevertheless, privacy challenges hamper the adoption of such services for sensitive data: Firstly, 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. Secondly, only being able to expose complete (authenticated) files may force users to expose overmuch information. The receivers, as well as the requirements, might be unknown at issue-time, and thus the issued data set does not exactly match those requirements. This mismatch 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 and redactable signature schemes. Proxy re-encryption provides us with the basis for end-to-end encrypted data-sharing, while redactable signatures enable to redact parts and selectively disclose only the remaining 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 conclude with an extensive performance evaluation of our instantiation.

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

⋆ Work done while the author was with Graz University of Technology.

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 [24]). 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 disclosure 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. [6] 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 receivers 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 paper 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 contain privacy-sensitive data. 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.

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 *unforgability*, we aim to conceal the plain text from unauthorized 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.

Finally, 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 [11] and DHS [18,19] 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 both a PC as well as a mobile phone.

Related Work. Proxy re-encryption, introduced by Blaze et al. [6], 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. [3] proposed the first strongly secure constructions, while follow-up work focused on stronger security notions [12, 30], performance improvements [16], and features such as forward secrecy [20], key-privacy [1], or lattice-based instantiations [13].

Attribute-based encryption (ABE) [25, 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) [7], 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 [8, 36] and graphs [29]. As a stronger privacy notion, transparency [8] 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. [22]. 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 [21]. We refer to [17] 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 [23] 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 [14,15] and the most prominent instantiations are Microsoft's U-Prove [33] and IBM's identity mixer [9,10]. However, as the plain data is required to compute the presentations, this operation must be performed in a sufficiently trusted environment.

In our previous work [26], we have identified the need for selective disclosure 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.

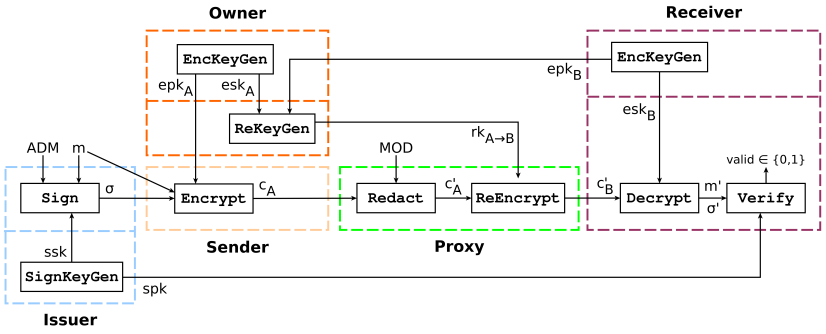

Fig. 1. Algorithms Performed by Actors (Dashed Lines Denote Trust Boundaries)

2 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.

Data Flow. As an informal overview, Figure 1 illustrates the model's algorithms 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 owner. 3) The *owner* is the entity, for which the data was originally encrypted. Initially, only this owner can decrypt the ciphertext. The owner may generate 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. Of course, multiple of these roles might be held by the same entity. For example, an owner signs her data (as issuer), uploads data (as sender), or accesses data she outsourced (as receiver).

Notation. In our following definitions, we adapt the syntax and notions inspired by standard definitions of PRE [2] and RSS [22].

Definition 1 (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.*

$\mathsf{SignKeyGen}(1^\kappa) \to (\mathsf{ssk}, \mathsf{spk})$**:** *On input of a security parameter κ, this probabilistic algorithm outputs a signature keypair* $(\mathsf{ssk}, \mathsf{spk})$.

$\mathsf{EncKeyGen}(1^\kappa) \to (\mathsf{esk}, \mathsf{epk})$**:** *On input of a security parameter κ, this probabilistic algorithm outputs an encryption keypair* $(\mathsf{esk}, \mathsf{epk})$.

$\mathsf{ReKeyGen}(\mathsf{esk}_A, \mathsf{epk}_B) \to \mathsf{rk}_{A\to B}$**:** *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 $\mathsf{rk}_{A\to B}$.*

$\mathsf{Sign}(\mathsf{ssk}, m, \mathsf{ADM}) \to \sigma$**:** *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.*

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

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

$\mathsf{Decrypt}(\mathsf{esk}_A, c_A) \rightarrow (m, \sigma)$**:** *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.*

$\mathsf{Redact}(c_A, \mathsf{MOD}) \rightarrow c'_A$**:** *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.*

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

Correctness. The correctness property requires that all honestly signed, encrypted, and possibly redacted and re-encrypted ciphertexts can be correctly decrypted and verified.

$$
\begin{array}{l|l|l}
\forall \kappa \in N, \forall(\mathsf{ssk}, \mathsf{spk}) \leftarrow \mathsf{SignKeyGen}(1^\kappa) & & \mathsf{Decrypt}(\mathsf{esk}_A, c_A) = (m, \sigma), \\
\forall(\mathsf{esk}_A, \mathsf{epk}_A) \leftarrow \mathsf{EncKeyGen}(1^\kappa) & & \mathsf{Decrypt}(\mathsf{esk}_B, c_B) = (m, \sigma), \\
\forall(\mathsf{esk}_B, \mathsf{epk}_B) \leftarrow \mathsf{EncKeyGen}(1^\kappa) & & \mathsf{Verify}(\mathsf{spk}, m, \sigma) = 1, \\
\forall \mathsf{rk}_{A\rightarrow B} \leftarrow \mathsf{ReKeyGen}(\mathsf{esk}_A, \mathsf{epk}_B) & & (m', \sigma') \leftarrow \mathsf{Decrypt}(\mathsf{esk}_A, c'_A), \\
\forall m, \forall \mathsf{ADM} \preceq m, \forall \sigma \leftarrow \mathsf{Sign}(\mathsf{ssk}, m, \mathsf{ADM}) & : & \mathsf{Verify}(\mathsf{spk}, m', \sigma') = 1, \\
\forall c_A \leftarrow \mathsf{Encrypt}(\mathsf{epk}_A, m, \sigma) & & (m', \sigma'') \leftarrow \mathsf{Decrypt}(\mathsf{esk}_B, c'_B), \\
\forall c_B \leftarrow \mathsf{ReEncrypt}(\mathsf{rk}_{A\rightarrow B}, c_A), & & \mathsf{Verify}(\mathsf{spk}, m', \sigma'') = 1, \\
\forall \mathsf{MOD} \overset{\mathsf{ADM}}{\preceq} m, \forall c'_A \leftarrow \mathsf{Redact}(c_A, \mathsf{MOD}) & & \text{with } m' \xleftarrow{\mathsf{MOD}} m_i. \\
\forall c'_B \leftarrow \mathsf{ReEncrypt}(\mathsf{rk}_{A\rightarrow B}, c'_A), & &
\end{array}
$$

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.

Add User Oracle, $\mathtt{AU}(i, t)$**:** Generates and tracks all key pairs for a user.

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

Sign Oracle, $\mathtt{SIG}(i, m, \mathsf{ADM})$**:** Signs messages for the challenge and honest users, and tracks all signatures.

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

Sign Encrypt Oracle, $\mathtt{SE}(i, j, m, \mathsf{ADM})$**:** Signs and encrypts messages for any user, and tracks all signatures.

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

Re-Encryption Key Generator Oracle, $\mathtt{RKG}(i, j)$**:** Generates keys for re-encryptions keys except for re-encryptions from the challenge user to a corrupted user.

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

Re-Encrypt Oracle, $\mathtt{RE}(i, j, c_j)$**:** Performs re-encryption of a ciphertext as long as the target user is not corrupted and the ciphertext is not derived from the challenge.

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

Decrypt Oracle, $\mathtt{D}(i, c_i)$**:** Decrypts a ciphertext as long as it is not derived from the challenge ciphertext.

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

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 ($\mathtt{SIG}$). Of course, we exclude signatures or their redactions that were obtained by adaptive queries to that signature oracle.

Experiment $\mathsf{Exp}^{\mathsf{Unf}}_{\mathsf{SEEDS},\mathcal{A}}(1^\kappa)$
$\mathtt{CH} \leftarrow \{0\}$, $SK[0] \leftarrow (\mathsf{ssk}^*, \mathsf{spk}^*) \leftarrow \mathsf{SignKeyGen}(1^\kappa)$, $\mathcal{O} \leftarrow \{\mathtt{SIG}\}$, $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{spk}^*)$
if $\mathsf{Verify}(\mathsf{spk}^*, m, \sigma) = 1$ and $m \notin \mathtt{Sigs}$, then **return** 1, **else return** 0

Experiment 1: Unforgeability Experiment for Signatures of SEEDS Schemes

Definition 2 (Unforgeability). *A* SEEDS *scheme is unforgeable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ε *such that*

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

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 $\mathsf{Exp}^{\mathsf{PP}}_{\mathsf{SEEDS},\mathcal{A}}(1^\kappa)$
$\mathtt{SK}[0] \leftarrow (\mathsf{ssk}^*, \mathsf{spk}^*) \leftarrow \mathsf{SignKeyGen}(1^\kappa)$, $\mathtt{EK}[0] \leftarrow (\mathsf{esk}^*, \mathsf{epk}^*) \leftarrow \mathsf{EncKeyGen}(1^\kappa)$
$\mathtt{CH} \leftarrow \{0\}$, $b \xleftarrow{R} \{0,1\}$, $\mathcal{O}_1 \leftarrow \{\mathtt{AU}, \mathtt{SE}, \mathtt{RKG}, \mathtt{RE}, \mathtt{D}\}$, $\mathcal{O}_2 \leftarrow \{\mathtt{SE}, \mathtt{RE}, \mathtt{D}\}$
$(m_0, \mathsf{ADM}_0, m_1, \mathsf{ADM}_1, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}_1}(\mathsf{spk}^*, \mathsf{epk}^*)$
if $m_0 \not\sim m_1$, or $\mathsf{ADM}_0 \not\sim \mathsf{ADM}_1$: **abort**
$\sigma \leftarrow \mathsf{Sign}(\mathsf{ssk}^*, m_b, \mathsf{ADM}_b)$, $c^* \leftarrow \mathsf{Encrypt}(\mathsf{epk}^*, m_b, \sigma)$, $b' \leftarrow \mathcal{A}^{\mathcal{O}_2}(\mathsf{st}, c^*)$
if $b = b'$, then **return** 1, **else return** 0

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

Definition 3 (Proxy Privacy). *A* SEEDS *scheme is proxy private, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that*

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

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

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

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

Experiment $\mathsf{Exp}^{\mathsf{RP}}_{\mathsf{SEEDS},\mathcal{A}}(1^\kappa)$
- $\mathtt{SK}[0] \leftarrow (\mathsf{ssk}^*, \mathsf{spk}^*) \leftarrow \mathsf{SignKeyGen}(1^\kappa)$
- $\mathtt{EK}[0] \leftarrow (\mathsf{esk}^*, \mathsf{epk}^*) \leftarrow \mathsf{EncKeyGen}(1^\kappa)$
- $\mathtt{CH} \leftarrow \{0\}$, $b \xleftarrow{R} \{0,1\}$
- $\mathcal{O} \leftarrow \{\mathtt{AU}, \mathtt{SIG}, \mathtt{LoRRedact}(\cdot\cdots\cdot, b)\}$
- $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{spk}^*, \mathsf{epk}^*)$
- **if** $b = b'$, then **return** 1
- **else return** 0

$\mathtt{LoRRedact}(i,j,m_0,\mathsf{MOD}_0,m_1,\mathsf{MOD}_1,\mathsf{ADM},b)$**:**
- **if** $i \notin \mathtt{CH}$ or $j \notin \mathtt{HU} \cup \mathtt{CU}$: **return** $\perp$
- $\mathsf{rk}_{A\to B} \leftarrow \mathsf{ReKeyGen}(\mathtt{EK}[i][0], \mathtt{EK}[j][1])$
- **for** both $c \in \{0,1\}$
  - $\sigma_c \leftarrow \mathsf{Sign}(\mathtt{SK}[i][0], m_c, \mathsf{ADM})$
  - $c_{A,c} \leftarrow \mathsf{Encrypt}(\mathtt{EK}[i][1], m_c, \sigma_c)$
  - $c'_{A,c} \leftarrow \mathsf{Redact}(c_{A,c}, \mathsf{MOD}_c)$
  - $c'_{B,c} \leftarrow \mathsf{ReEncrypt}(\mathsf{rk}_{A\to B}, c'_{A,c})$
  - $(m'_c, \sigma'_c) \leftarrow \mathsf{Decrypt}(\mathtt{EK}[j][0], c'_{B,c})$
- **if** $m'_0 = m'_1$: **return** $c'_{B,b}$
- **else**: **return** $\perp$

Experiment 3: Receiver Privacy Experiment for SEEDS Schemes

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

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

Experiment 4: Transparency Experiment for Ciphertexts of SEEDS Schemes

Definition 5 (Transparency). *A* SEEDS *scheme is transparent, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ε such that*

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

3 Modular Instantiation

Scheme 1 instantiates our model by building on generic cryptographic mechanisms, most prominently proxy re-encryption and redactable signatures, which can be instantiated with various underlying schemes.

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, C), and incorporate this commitment C 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 C 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 C is added to the signature but must not be redacted, so it is internally added to ADM.

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. [22, 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.

As public parameters, fix a proxy re-encryption scheme PRE, a symmetric encryption scheme S, a commitment scheme C with fixed parameters $cpp \leftarrow \mathsf{C.KeyGen}(1^\kappa)$, and a redactable signature scheme RSS.

$\underline{\mathsf{SignKeyGen}}(1^\kappa) \to (\mathsf{ssk}, \mathsf{spk})$: **return** $\mathsf{RSS.KeyGen}(1^\kappa)$
$\underline{\mathsf{EncKeyGen}}(1^\kappa) \to (\mathsf{esk}, \mathsf{epk})$: **return** $\mathsf{PRE.KeyGen}(1^\kappa)$
$\underline{\mathsf{ReKeyGen}}(\mathsf{esk}_A, \mathsf{epk}_B) \to \mathsf{rk}_{A\to B}$: **return** $\mathsf{PRE.ReKeyGen}(\mathsf{esk}_A, \mathsf{epk}_B)$

$\underline{\mathsf{Sign}}(\mathsf{ssk}, m, \mathsf{ADM}) \to \sigma$:
- $k \leftarrow \mathsf{S.KeyGen}(1^\kappa)$
- $c \leftarrow \{\mathsf{S.Enc}(k, m_i) \mid m_i \in M\}$
- $c_{\mathsf{ADM}} \leftarrow \{c_i \in c \mid m_i \in \mathsf{ADM}\}$
- $(C, O) \leftarrow \mathsf{C.Com}(k)$
- $(\cdot, \hat{\sigma}) \leftarrow \mathsf{RSS.Sign}(\mathsf{ssk}, \{C\} \cup c, \{C\} \cup c_{\mathsf{ADM}})$
- **return** $\sigma \leftarrow (O, C, c, \hat{\sigma})$

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

$\underline{\mathsf{Encrypt}}(\mathsf{epk}_A, m, \sigma) \to c_A$:
- Parse σ as $(O, C, c, \hat{\sigma})$
- $O_A \leftarrow \mathsf{PRE.Enc}^2(\mathsf{epk}_A, O)$
- **return** $c_A \leftarrow (O_A, C, c, \hat{\sigma})$

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

$\underline{\mathsf{Redact}}(c_A, \mathsf{MOD}) \to c'_A$:
- Parse c_A as $(O_A, C, c, \hat{\sigma})$
- $(\{C\} \cup c', \hat{\sigma}') \leftarrow$
  - $\mathsf{RSS.Redact}(\{C\} \cup c, \hat{\sigma}, \mathsf{MOD})$
- **return** $c'_A \leftarrow (O_A, C, c', \hat{\sigma}')$

$\underline{\mathsf{ReEncrypt}}(\mathsf{rk}_{A\to B}, c_A) \to c_B$:
- Parse c_A as $(O_A, C, c, \hat{\sigma})$
- $O_B \leftarrow \mathsf{PRE.ReEnc}(\mathsf{rk}_{A\to B}, O_A)$
- **return** $c_B \leftarrow (O_B, C, c, \hat{\sigma})$

Scheme 1: Modular Instantiation

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

The proof is given in Appendix A.

4 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 200kB to 50 high-definition x-ray scans à 10MB.

Table 1. Cryptographic Building Blocks for Our Three Implementations

	Impl. 1: Sets & CL	**Impl. 2: Sets & DHS**	**Impl. 3: Lists & CL**
PRE	Chow et al. [16], 3072bit		
S	AES-CBC, 128bit		
C	Hash Commitment (SHA3), 256bit		
Hash	SHA3, 256bit		
RSS	DPSS [22, Scheme 1]	DPSS [22, Scheme 1]	DPSS [22, Scheme 2]
Accu.	CL [11], 3072bit	DHS [19, Scheme 3], 384bit	CL [11], 3072bit
DSS	RSA, 3072bit	ECDSA, 256bit	RSA, 3072bit

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 [5] for factoring-based and symmetric-key primitives. The groups for pairing-based curves are chosen following recent recommendations [4, 31]. These implementations were developed for the Java platform using the IAIK-JCE and ECCelerate libraries[3]. We selected the accumulators based on the comparison by Derler et al. [18].

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 cryptographic operations might be performed by less powerful mobile phones. Therefore, we performed the benchmarks on two platforms: a PC as well as an Android mobile phone. Table 2 summarizes the execution times of the different implementations for both 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 500MB test on the phone, as memory usage is limited to 192MB for apps on our Google Pixel 2.

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. Impl. 1 provides the best overall performance for verification. For the first implementation, we use an RSS scheme for sets [22, Scheme 1] with CL accumulators [11], where we hash the

[3] https://jce.iaik.tugraz.at/

Table 2. Execution Times (in Milliseconds) of Three Implementations for Scheme 1 (Dec* denotes decryption without additional signature verification)

		Impl. 1: Sets & CL						**Impl. 2: Sets & DHS**						**Impl. 3: Lists & CL**					
#	size	*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)																			
10x	1kB	26	1	12	1	1	13	13	2	<1	2	1	47	37	1	36	1	1	18
25x	1kB	54	1	28	1	1	28	19	2	<1	1	1	89	97	1	129	1	1	43
100x	1kB	179	1	97	1	1	89	63	2	<1	1	1	302	852	1	1591	1	1	265
5x	200kB	23	1	6	1	3	16	12	1	<1	1	4	38	28	1	16	1	3	18
25x	1MB	142	1	39	1	40	85	101	1	<1	1	33	143	189	1	151	1	33	111
50x	10MB	1871	1	393	1	749	1100	1777	1	<1	1	752	1216	2310	1	905	1	733	1285
Mobile Phone (Google Pixel 2)																			
10x	1kB	198	13	112	7	21	121	716	13	<1	6	20	2327	212	11	204	6	19	153
25x	1kB	360	12	199	6	22	211	1171	12	<1	6	21	4649	451	12	623	6	20	258
100x	1kB	1194	12	695	6	24	607	3578	12	1	6	27	15982	2631	12	6835	6	27	973
5x	200kB	177	12	73	6	28	106	657	12	<1	6	28	1896	188	12	92	6	24	110
25x	1MB	777	12	370	6	111	480	1654	12	<1	7	111	4936	893	11	808	6	111	533

message parts before signing. With this accumulator, it is possible to optimize the implementation, as described in [22], 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 [22, Scheme 1] but move towards elliptic curves by instantiating it with ECDSA signatures and DHS accumulators [19, Scheme 3] (extended version of [18]), 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, 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, phones seem to struggle with the implementation of this accumulator, resulting in far worse times than the otherwise observed slowdown compared with the PC. 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 [22, 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. [22] 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 again allows for an optimization [22] around batch witnesses and verifications. This optimization also reduces the RSS signature size from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

5 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. Finally, we evaluated the performance characteristics of three implementations to highlight the practical usefulness of our modular construction and model as a whole.

Acknowledgments. This work was supported by the H2020 EU project CREDENTIAL under grant agreement number 653454.

References

1. Ateniese, G., Benson, K., Hohenberger, S.: Key-private proxy re-encryption. In: CT-RSA. LNCS, vol. 5473, pp. 279–294. Springer (2009)
2. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. In: NDSS. The Internet Society (2005)
3. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. **9**(1), 1–30 (2006)
4. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. J. Cryptology **32**(4), 1298–1336 (2019)
5. Barker, E.: SP 800-57. Recommendation for Key Management, Part 1: General (Rev 4). Tech. rep., National Institute of Standards & Technology (2016)
6. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: EUROCRYPT. LNCS, vol. 1403, pp. 127–144. Springer (1998)
7. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: TCC. LNCS, vol. 6597, pp. 253–273. Springer (2011)
8. Brzuska, C., Busch, H., Dagdelen, Ö., Fischlin, M., Franz, M., Katzenbeisser, S., Manulis, M., Onete, C., Peter, A., Poettering, B., Schröder, D.: Redactable signatures for tree-structured data: Definitions and constructions. In: ACNS. LNCS, vol. 6123, pp. 87–104 (2010)

9. Camenisch, J., Herreweghen, E.V.: Design and implementation of the *idemix* anonymous credential system. In: ACM CCS. pp. 21–30. ACM (2002)
10. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: EUROCRYPT. LNCS, vol. 2045, pp. 93–118. Springer (2001)
11. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: CRYPTO. LNCS, vol. 2442, pp. 61–76. Springer (2002)
12. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: ACM CCS. pp. 185–194. ACM (2007)
13. Chandran, N., Chase, M., Liu, F., Nishimaki, R., Xagawa, K.: Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices. In: PKC. LNCS, vol. 8383, pp. 95–112. Springer (2014)
14. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–88 (1981)
15. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985)
16. Chow, S.S.M., Weng, J., Yang, Y., Deng, R.H.: Efficient unidirectional proxy re-encryption. In: AFRICACRYPT. LNCS, vol. 6055, pp. 316–332. Springer (2010)
17. Demirel, D., Derler, D., Hanser, C., Pöhls, H.C., Slamanig, D., Traverso, G.: PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Tech. rep., H2020 PRISMACLOUD (2015)
18. Derler, D., Hanser, C., Slamanig, D.: Revisiting cryptographic accumulators, additional properties and relations to other primitives. In: CT-RSA. LNCS, vol. 9048, pp. 127–144. Springer (2015)
19. Derler, D., Hanser, C., Slamanig, D.: Revisiting cryptographic accumulators, additional properties and relations to other primitives. IACR ePrint **2015**, 87 (2015)
20. Derler, D., Krenn, S., Lorünser, T., Ramacher, S., Slamanig, D., Striecks, C.: Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. In: PKC (1). LNCS, vol. 10769, pp. 219–250. Springer (2018)
21. Derler, D., Krenn, S., Slamanig, D.: Signer-anonymous designated-verifier redactable signatures for cloud-based data sharing. In: CANS. LNCS, vol. 10052, pp. 211–227 (2016)
22. Derler, D., Pöhls, H.C., Samelin, K., Slamanig, D.: A general framework for redactable signatures and new constructions. In: ICISC. LNCS, vol. 9558, pp. 3–19. Springer (2015)
23. Derler, D., Ramacher, S., Slamanig, D.: Homomorphic proxy re-authenticators and applications to verifiable multi-user data aggregation. In: Financial Cryptography. LNCS, vol. 10322, pp. 124–142. Springer (2017)
24. European Commission: 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** (May 2016)
25. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM CCS. pp. 89–98. ACM (2006)
26. Hörandner, F., Krenn, S., Migliavacca, A., Thiemer, F., Zwattendorfer, B.: CREDENTIAL: A framework for privacy-preserving cloud-based data sharing. In: ARES. pp. 742–749. IEEE Computer Society (2016)
27. Johnson, R., Molnar, D., Song, D.X., Wagner, D.A.: Homomorphic signature schemes. In: CT-RSA. LNCS, vol. 2271, pp. 244–262. Springer (2002)

28. Krenn, S., Lorünser, T., Salzer, A., Striecks, C.: Towards attribute-based credentials in the cloud. In: CANS. LNCS, vol. 11261, pp. 179–202. Springer (2017)
29. Kundu, A., Bertino, E.: Privacy-preserving authentication of trees and graphs. Int. J. Inf. Sec. **12**(6), 467–494 (2013)
30. Libert, B., Vergnaud, D.: Unidirectional chosen-ciphertext secure proxy re-encryption. In: PKC. LNCS, vol. 4939, pp. 360–379. Springer (2008)
31. Menezes, A., Sarkar, P., Singh, S.: Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In: Mycrypt. LNCS, vol. 10311, pp. 83–108. Springer (2016)
32. Nguyen, L.: Accumulators from bilinear pairings and applications. In: CT-RSA. LNCS, vol. 3376, pp. 275–292. Springer (2005)
33. Paquin, C., Zaverucha, G.: U-prove cryptographic specification v1.1 (revision 3). Tech. rep., Microsoft (Dec 2013)
34. Pirretti, M., Traynor, P., McDaniel, P.D., Waters, B.: Secure attribute-based systems. In: ACM CCS. pp. 99–112. ACM (2006)
35. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT. LNCS, vol. 3494, pp. 457–473. Springer (2005)
36. Samelin, K., Pöhls, H.C., Bilzhause, A., Posegga, J., de Meer, H.: Redactable signatures for independent removal of structure and content. In: ISPEC. LNCS, vol. 7232, pp. 17–33. Springer (2012)
37. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: ICISC. LNCS, vol. 2288, pp. 285–304. Springer (2001)
38. Weng, J., Deng, R.H., Ding, X., Chu, C., Lai, J.: Conditional proxy re-encryption secure against chosen-ciphertext attack. In: AsiaCCS. pp. 322–332. ACM (2009)

A Proof of Theorem 1

We prove Theorem 1 by proving Lemma 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. *If* RSS *is unforgeable and* C *is binding, then Scheme 1 is unforgeable.*

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 $\mathtt{SIG}$.

$\mathtt{SIG}(i, m, \mathsf{ADM})$:
- **if** $i \notin \mathtt{CH} \cup \mathtt{HU}$ or $\mathtt{SK}[i][0] = \bot$: return $\bot$
- $\sigma \leftarrow \mathsf{Sign}(\mathtt{SK}[i][0], m, \mathsf{ADM})$
- $\boxed{\text{Parse } \sigma \text{ as } (O, C, c, \hat{\sigma})}$
- $\mathtt{Sigs} \leftarrow \mathtt{Sigs}$
  $\cup \{m' \mid \forall \mathsf{MOD} \overset{\mathsf{ADM}}{\preceq} m \; \forall m' \xleftarrow{\mathsf{MOD}} m\}$
- $\boxed{\mathtt{Coms} \leftarrow \mathtt{Coms} \cup \{C\}}$
- **return** (m, σ)

Game 1:
- $(\mathsf{ssk}^*, \mathsf{spk}^*) \leftarrow \mathsf{SignKeyGen}(1^\kappa)$
- $\mathtt{CH} \leftarrow \{0\}$, $SK[0] \leftarrow (\mathsf{ssk}^*, \mathsf{spk}^*)$
- $\mathcal{O} \leftarrow \{\mathtt{SIG}\}$
- $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{spk}^*)$
- $\boxed{\text{Parse } \sigma \text{ as } (O, C, c, \hat{\sigma})}$
- $\boxed{\textbf{if } C \in \mathtt{Coms}\text{: } \textbf{return } 0}$
- **if** $\mathsf{Verify}(\mathsf{spk}^*, m, \sigma) = 1$ and $m \notin \mathtt{Sigs}$, then **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 $\{C\} \cup c$ was generated by $\mathtt{SIG}$. We denote this failure event as F, thus $|\Pr[S_0] - \Pr[S_1]| \leq \Pr[F]$. In this case, since C, 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 C, different keys can only be recovered with different opening informations O_1 and O_2. To achieve this, the adversary would have to break the binding property of C, therefore $\Pr[F] = \epsilon_{\mathsf{C}}^{Bind}(\kappa)$.

Finally, we build an efficient adversary $\mathcal{B}$ from an adversary $\mathcal{A}$ winning Game 1 for the unforgeability of RSS in $\mathcal{R}_{\mathsf{RSS}}^{Unf} \rightarrow G_1$. We can simulate $\mathtt{SIG}$ 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 $(O, C, c, \hat{\sigma})$ and forward $\{C\} \cup c, \hat{\sigma}$ as a forgery. Therefore, $\Pr[S_1] = \epsilon_{\mathsf{RSS}}^{Unf}(\kappa)$, resulting in $\Pr[S_0] = \epsilon_{\mathsf{C}}^{Bind}(\kappa) + \epsilon_{\mathsf{RSS}}^{Unf}(\kappa)$, which is negligible.

Reduction $\mathcal{R}_{\mathsf{RSS}}^{Unf} \rightarrow G_1(\mathsf{pk})$:
$(m, \sigma) \leftarrow \mathcal{A}^{\mathtt{SIG}}(\mathsf{pk})$
Parse σ as $(O, C, C, \hat{\sigma})$
return $(\{C\} \cup c, \hat{\sigma})$

$\mathtt{SIG}(0, m, \mathsf{ADM})$**:**
$k \leftarrow \mathsf{S.KeyGen}(1^\kappa)$
$c \leftarrow \{\mathsf{S.Enc}(k, m_i) \mid m_i \in M\}$
$c_{\mathsf{ADM}} \leftarrow \{c_i \mid c_i \in c, m_i \in \mathsf{ADM}\}$
$\sigma \leftarrow (O, C, c, \mathcal{O}^{Sign}(\underline{\mathsf{sk}}, \{C\} \cup c, \{C\} \cup c_{\mathsf{ADM}}))$
$\mathtt{Sigs} \leftarrow \mathtt{Sigs} \cup \{m' \mid \forall \mathsf{MOD} \overset{\mathsf{ADM}}{\preceq} m \; \forall m' \xleftarrow{\mathsf{MOD}} m\}$
$\mathtt{Coms} \leftarrow \mathtt{Coms} \cup \{C\}$
return (m', σ)

Lemma 2. *If the* PRE *is IND-RCCA-2 secure,* C *is hiding,* S *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 oracle. Therefore, we adapt $\mathtt{SE}$ as $\mathtt{SIG}$ in Lemma 1 to track the generated commitments, $\boxed{\mathtt{Coms} \leftarrow \mathtt{Coms} \cup \{C\}}$. Also, we adapt $\mathtt{D}$ to $\boxed{\text{parse } c_i \text{ as } (O_A, C, c, \hat{\sigma})}$ and $\boxed{\textbf{if } C \notin \mathtt{Coms}\textbf{: return } \perp}$.

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

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

$\mathsf{Encrypt}$**:**
Parse σ as $(O, C, c, \hat{\sigma})$
$\boxed{r \xleftarrow{R} Domain(\mathsf{C}_O)}$
$O_A \leftarrow \mathsf{PRE.Enc}^2(\mathsf{epk}_A, \boxed{\Gamma})$
$\boxed{\mathtt{Map} \leftarrow \mathtt{Map} \cup \{(O_A, C, O)\}}$
return $c_A \leftarrow (O_A, C, c, \hat{\sigma})$.

$\mathtt{RE}(i, j, k, c_j)$**:**
Parse c_j as $(O_A, C, c, \hat{\sigma})$
Look up $(O_A, C, O) \in \mathtt{Map}$
if not contained: run $\mathtt{RE}$ unmodified
else
$O_B \leftarrow \mathsf{PRE.ReEnc}(\mathsf{rk}_{i \rightarrow j}, O_A)$
$\mathtt{Map} \leftarrow \mathtt{Map} \cup \{(O_B, C, O)\}$
return $(O_B, C, c, \hat{\sigma})$

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

Transition $1 \Rightarrow 2$**:** From a distinguisher $\mathcal{D}^{1\to 2}$, 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 $\boxed{\mathtt{r}}$ uniformly at random from the domain of openings of C 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 $\mathtt{RE}$ oracle calls the challenger's $\mathtt{RE}$ oracle instead of PRE.ReEnc. Consequently, $|\Pr[S_1] - \Pr[S_2]| \leq \epsilon_{\mathsf{PRE}}^{IND-RCCA-2}(\kappa)$.

Game 3: For the signature contained in the challenge ciphertext, we commit to a random value, i.e., we set $\boxed{r \xleftarrow{R} \mathit{Domain}(\mathsf{S}_k)}$ and $(C, O) \leftarrow \mathsf{C.Com}(\boxed{\mathtt{r}})$.

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

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. $\boxed{\text{for } i \in [1..|M|]\text{: } r_i \xleftarrow{R} \mathit{Domain}(m)}$ and $c \leftarrow \{\mathsf{S.Enc}(k, \boxed{r_i}) \mid m_i \in m\}$.

Transition $3 \Rightarrow 4$**:** A distinguisher $\mathcal{D}^{3\to 4}$ is a (hybrid) IND-CPA adversary against S. 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, $|\Pr[S_4] - \Pr[S_3]| \leq |m| \cdot \epsilon_{\mathsf{S}}^{IND-CPA}(\kappa)$, 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:

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

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 $\mathcal{R}_{\mathsf{RSS}}^{Priv} \rightarrow_{\mathsf{SEEDS}}^{RP} (\mathsf{pk})$**:**
$(\mathsf{esk}^*, \mathsf{epk}^*) \leftarrow \mathsf{SEEDS.EncKeyGen}(1^\kappa)$
$\mathtt{CH} \leftarrow \{0\}$
$\mathtt{EK}[0] \leftarrow (\mathsf{esk}^*, \mathsf{epk}^*)$, $\mathtt{SK}[0] \leftarrow (\bot, \mathsf{pk})$
$\mathcal{O} \leftarrow \{\mathtt{AU}, \mathtt{SIG}, \mathtt{LoRRedact}(\cdot\cdot\cdot\cdot\cdot\cdot\cdot, b)\}$
return $b^* \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}, \mathsf{epk}^*)$

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

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

$\mathtt{LoRRedact}(i,j,m_0,\mathsf{MOD}_0,m_1,\mathsf{MOD}_1,\mathsf{ADM},b)$**:** For $i \neq 0$ or $j \notin \mathtt{HU} \cup \mathtt{CU}$ everything is computed honestly, otherwise we run:
$\mathsf{rk} \leftarrow \mathsf{ReKeyGen}(\mathtt{EK}[i][0], \mathtt{EK}[j][1])$
$k \leftarrow \mathsf{S.KeyGen}(1^\kappa; \omega)$
$(C, O) \leftarrow \mathsf{C.Com}(k)$
$O_B \leftarrow \mathsf{PRE.ReEnc}(\mathsf{rk}, \mathsf{PRE.Enc}(\mathtt{EK}[i][1], O))$
for both $c \in \{0, 1\}$:
$c_c \leftarrow \{\mathsf{S.Enc}(k, m_i) \mid m_i \in m_c\}$
$c_{\mathsf{ADM},c} \leftarrow \{c_i \mid c_i \in c_c, m_i \in \mathsf{ADM}\}$
$X \leftarrow \mathcal{O}^{\mathtt{LoRRedact}}(\underline{\mathsf{sk}}, \underline{\mathsf{pk}}, (\{C\} \cup c_c, \{C\} \cup c_{\mathsf{ADM},c}, \mathsf{MOD}_c)_{c \in \{0,1\}}, \underline{b})$
if $X = \bot$: return $\bot$
Parse X as $(\{C\} \cup c_{b'}, \hat{\sigma}'_b)$
return $(O_B, C, c'_b, \hat{\sigma}'_b)$

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 $\mathcal{R}_{\mathsf{RSS}}^{Trans} \rightarrow_{\mathsf{SEEDS}}^{Trans} (\mathsf{pk})$**:**
$(\mathsf{esk}^*, \mathsf{epk}^*) \leftarrow \mathsf{SEEDS.EncKeyGen}(1^\kappa)$
$\mathtt{CH} \leftarrow \{0\}$
$\mathtt{EK}[0] \leftarrow (\mathsf{esk}^*, \mathsf{epk}^*)$, $\mathtt{SK}[0] \leftarrow (\bot, \mathsf{pk})$
$\mathcal{O} \leftarrow \{\mathtt{AU}, \mathtt{SIG}, \mathtt{RedactOrNot}\}$
return $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{esk}^*, \mathsf{epk}^*, \mathsf{pk})$

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

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

$\mathtt{RedactOrNot}(i, m, \mathsf{MOD}, \mathsf{ADM}, b)$**:** For $i \neq 0$ everything is computed honestly, otherwise we run the following:
$k \leftarrow \mathsf{S.KeyGen}(1^\kappa; \omega)$
$(C, O) \leftarrow \mathsf{C.Com}(k)$
$O_A \leftarrow \mathsf{PRE.Enc}(\mathtt{EK}[i][1], O)$
$c \leftarrow \{\mathsf{S.Enc}(k, m_i) \mid m_i \in m\}$
$c_{\mathsf{ADM}} \leftarrow \{c_i \mid c_i \in c, m_i \in \mathsf{ADM}\}$
$(\{C\} \cup c', \hat{\sigma}') \leftarrow \mathcal{O}^{Sign/Redact}(\underline{\mathsf{sk}}, \underline{\mathsf{pk}}, \{C\} \cup c, \mathsf{MOD}, \{C\} \cup c_{\mathsf{ADM}}, \underline{b})$
return $c'_{A,b} \leftarrow (O_A, C, c', \hat{\sigma}')$

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 $\mathcal{O}^{Sign/Redact}$ to obtain the signature. Finally, it outputs a consistent ciphertext, hence, $\mathcal{B}$ wins with the same probability as $\mathcal{A}$.