Short Redactable Signatures for Strings Using Random Tree

Ee-Chien Chang Chee Liang Lim Jia Xu

Abstract. A redactable signature scheme for a string of objects supports verification even if multiple substrings are removed from the original string. It is important that the redacted string and its signature do not reveal anything about the content of the removed substrings. Existing schemes completely or partially leak a piece of information: the lengths of the removed substrings. Such length information could be crucial for many applications, especially when the removed substring has low entropy. We propose a scheme that can hide the length. Our scheme consists of two components. The first component $\mathcal{H}$, which is a "collision resistant" hash, maps a string to an unordered set, whereby existing schemes on unordered sets can then be applied. However, a sequence of random numbers has to be explicitly stored and thus it produces a large signature of size at least (mk)-bits where m is the number of objects and k is the size of a key sufficiently large for cryptographic operations. The second component uses RGGM tree, a variant of GGM tree, to generate the pseudo random numbers from a short seed, expected to be of size $O(k + tk \log m)$ where t is the number of removed substrings. Unlike GGM tree, the structure of the proposed RGGM tree is random. By an intriguing statistical property of the random tree, the redacted tree does not reveal the lengths of the substrings removed. The hash function $\mathcal{H}$ and the RGGM tree can be of independent interests.

Key words: Redactable Signature Scheme, Random Tree, Privacy

1 Introduction

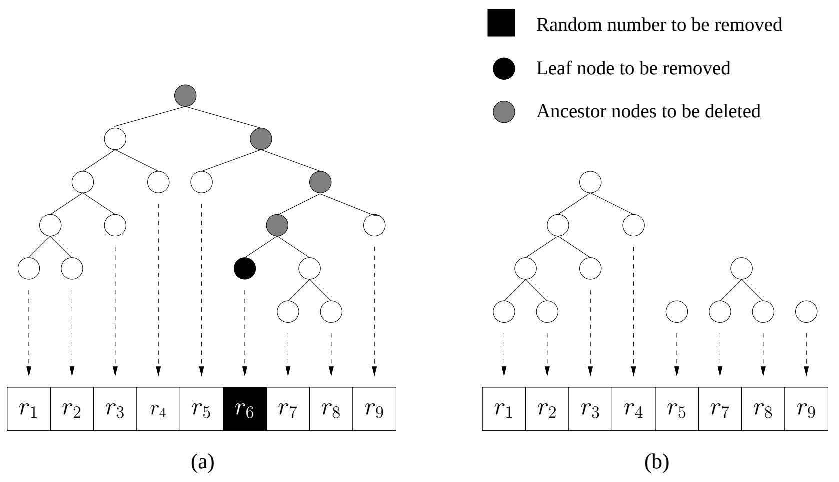
We are interested in a signature scheme for strings of objects whereby the authenticity of a string can be verified even if it is redacted, that is, some substrings have been removed. Let $\mathbf{x} = x_1 x_2 \ldots x_m$ be a string, for example a text document where each object can be a character or a word, or an audio file where each object is a sample. The string $\mathbf{x}$ is signed by the authority and both the string and its signature $(\mathbf{x}, \mathbf{s})$ are passed to another party, say Alice. Alice wants to show Bob $\mathbf{x}$ but Bob is not authorized to view certain parts of the string, say $x_2 x_3 x_4$ and x_7. Thus, Alice shows Bob $\widetilde{\mathbf{x}} = x_1 \diamond x_5 x_6 \diamond x_8 \ldots x_m$ where each $\diamond$ indicates the location of a removed substring. On the other hand, Bob may want to verify the authenticity of $\widetilde{\mathbf{x}}$. A redactable signature scheme allows Alice to produce a valid signature $\widetilde{\mathbf{s}}$ for the redacted string $\widetilde{\mathbf{x}}$, even if Alice does not have the authority's secret key. From the new signature $\widetilde{\mathbf{s}}$, Bob can then verify that $\widetilde{\mathbf{x}}$ is indeed a redacted version of a string signed by the authority.

Unlike the usual signature schemes, redactable signature scheme has additional requirement on privacy: information of the removed strings should be hidden. In this paper, we consider the stringent requirement that, Bob could not obtain any information of any removed substring, except the fact that a non-empty substring has been removed at each location $\diamond$. This simple requirement turns out to be difficult to achieve. Existing schemes are unable to completely hide a piece of usually important information: the length of each removed substring. Note that information on length could be crucial if the substring has low entropy. For example, if the substring is either "Approved" or "Not Approved", then its length reveals everything. There are scenarios and applications where the lengths of the removed strings should not be hidden. As noted by Johnson et al. [7], semantic attack could be possible in some scenarios if the length information is hidden. On the other side, there are also applications where the fact that a string has been redacted must be hidden. Our scheme can be modified to cater for the above scenarios. The redactable signature scheme proposed by Johnson et al. [7] employs a Merkle tree [9] and a GGM tree [5] to generate a short signature. However, it is easy to derive the length from the structures of the redacted Merkle and GGM trees. A straightforward modification by introducing randomness into the tree structure also does not hide the length completely. Schemes by Johnson et al. [7] (set-homomorphic signatures) and Miyazaki et al. [11] are designed for unordered sets and are not applicable for a string. A way to extend their schemes to strings is by assigning a sequence of increasing random numbers to the objects [11]. However, this leads to large signatures since the random numbers have to be explicitly stored, and more importantly, it is insecure since the gaps in the sequence reveal some information about the number of removed objects.

In this paper, we propose a scheme that can hide the lengths of the removed substrings. Our scheme incorporates two components: a hash, and a random tree with a hiding property. We first give a scheme `S4` using the first component, and then another scheme `S5` with both components. The first component hashes a string of objects to an unordered set. For the unordered set, existing redactable schemes [11, 7] on unordered sets can be applied. The scheme `S4` satisfies the requirements on unforgeability and privacy preserving under reasonable cryptographic assumptions. However, it produces a large signature. Essentially, the main portion of the signature is a sequence of random numbers $\langle r_1, r_2, \ldots, r_m \rangle$, where each r_i is associated with the i-th object in the string.

The goal of the second component is to reduce the signature size by replacing the r_i's with a sequence of pseudo random numbers, generated from a small seed t. If a substring is removed, the corresponding random numbers have to be removed accordingly. Thus, generating the random numbers iteratively starting from the seed violates privacy, since the seed t reveals all the random numbers.

We employ a variant of GGM binary tree to generate the r_i's in a top-down manner, where the r_i's are at the leaves, and the seed t is at the root. Unlike the GGM tree which is balanced, we use a random binary tree where the *structure* of the binary tree is random. After a substring is removed, the associated leaves and all their ancestors are to be removed, resulting in a collection of subtrees (Figure 1). The roots of the subtrees collectively form the new seed $\widetilde{t}$ for the redacted r_i's. Note that from the structures of the subtrees, an adversary might still derive some information of the length of a removed substring. Our main observation is that, by choosing an appropriate tree generation algorithm, the structure of the subtrees reveals nothing about the size of the original tree. Consider a game between Alice and Bob. Suppose Alice randomly picks a binary tree and it is equal likely that the tree contains 1000 leaves or 9 leaves. Now Alice redacts the tree by removing one substring and only 8 leaves are left. From the structure of the remaining subtrees (for example Figure 1(b)), Bob tries to guess the size of the original tree. Now, if Alice employs a tree generation algorithm with the hiding property, Bob can not succeed with probability more than 0.5. This hiding property is due to the independence between the sizes of subtrees rooted at two sibling nodes, without knowing the sum of the two sizes.



Random number to be removed
Leaf node to be removed
Ancestor nodes to be deleted
r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9
(a)
r_1 r_2 r_3 r_4 r_5 r_7 r_8 r_9
(b)

Fig. 1. Redacting the tree in (a) by removing r_6, gives rise to the redacted tree (b).

Contribution and Organization.
1. We propose a "collision resistant" hash $\mathcal{H}$ that maps strings to unordered sets. From $\mathcal{H}$ we obtain `S4`, a redactable signature scheme for strings. Unlike previously known methods, `S4` is able to hide the lengths of the removed substrings. We show that `S4` is secure against chosen message attack (Theorem 2) and privacy preserving (Theorem 3) under resonable assumptions, which are weaker than random oracle assumption. However, the signature size is large. It consists of $km + kt + \kappa$ bits, where κ is the size of the signature produced by a known redactable signature scheme for unordered sets, m is the number of objects in the redacted string, t is the number of substrings removed, and k is a security parameter (e.g. $k = 1024$).

2. We observe a hiding property of a random tree (Theorem 4). Based on the observation, we propose RGGM, a pseudo random number generator which can be viewed as a randomized version of GGM [5]. If multiple substrings of pseudo random numbers are to be removed, we can efficiently find a new seed that generates the retained numbers, and yet it is computationally difficult to derive the content and length of each removed substring from the new seed, except the locations of the removed substrings.
3. We propose `S5` by incorporating RGGM into `S4`. The expected size of the signature is in $\kappa + O(k + kt \log m)$. `S5` is secure against chosen message attack (Corollary 5 in Appendix E) and privacy preserving (Corollary 6 in Appendix E).

2 Related Work

Johnson et al. [7] introduced redactable signature schemes which enable verification of a redacted signed document. Signature scheme with similar property has also been proposed for XML documents [13], where the redaction operation is to remove XML nodes. Redactable signatures are examples of homomorphic signatures which are introduced by Rivest in his talks on "Two New Signature Schemes" [12] and formalized by Johnson et al. [7]. Micali et al. [10] gave a transitive signature scheme as the first construction of homomorphic signatures. They also asked for other possible "signature algebras". The notions on homomorphic signatures can be traced back to incremental cryptography, introduced by Bellare, Goldreich and Goldwasser [2, 3]. Recently, Ateniese et al. [1] introduced sanitizable signature scheme [8, 6, 14, 11] allowing a semi-trusted censor modifies the signed documents in a limited and controlled way.

The redactable signature scheme on strings is closely related to directed transitive signature scheme [10, 15]: It is possible to convert a directed transitive signature scheme to a redactable signature scheme on strings. However, existing directed transitive signature schemes do not provide privacy in the sense that the resulting signatures reveal some information about the removed substrings.

3 Formulation and Background

Johnson et al.[7] gave definitions on homomorhpic signature schemes and their security for binary operators. It can be easily adapted for binary relations.

A string is a sequence of objects from an object space (or alphabet) $\mathbb{O}$. For example, $\mathbb{O}$ can be the set of ASCII characters, collection of words, or audio samples, etc. After a few substrings are removed from $\mathbf{x}$, the string $\mathbf{x}$ may break into substrings, say $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_u$. The redacted string $(\hat{\mathbf{x}}, \mathbf{e})$, which we call *annotated string*[1], is represented by the string $\hat{\mathbf{x}} = \mathbf{x}_1 \| \mathbf{x}_2 \| \ldots \| \mathbf{x}_u$ and an *annotation* $\mathbf{e} = \langle m, b_1, b_2, \ldots, b_v \rangle$ where $\|$ denotes concatenation, b_i's is a strictly increasing sequence indicating the locations of the removed substrings, m is the number of objects in $\widetilde{\mathbf{x}}$, and $v \in \{u-1, u, u+1\}$. For each i, b_i indicates that a non-empty

[1] A string with an annotation which specifies the locations of redactions.

substring has been removed in between the b_i-th and $(1 + b_i)$-th locations. If $b_1 = 0$ or $b_v = m$, this indicates that a non-empty substring has been removed at the beginning or end of the string respectively. For example, $(abcda, \langle 5, 0, 3\rangle)$ is a redacted string of the original $xxxabcyyyda$. For convenient, we sometimes use a sequence of objects like $\langle x_1, x_2, x_3, \ldots, x_m\rangle$ and a string of objects like $x_1x_2x_3 \ldots x_m$ interchangeably.

Let us define a binary relation $\succ$ between annotated strings. Given two annotated strings $\widetilde{\mathbf{x}}_1 = (\mathbf{x}_1, \mathbf{e}_1)$ and $\widetilde{\mathbf{x}}_2 = (\mathbf{x}_2, \mathbf{e}_2)$, we say $\widetilde{\mathbf{x}}_1 \succ \widetilde{\mathbf{x}}_2$, if 1) $\mathbf{x}_2$ can be obtained from $\mathbf{x}_1$ by removing a non-empty substring in $\mathbf{x}_1$, and the $\mathbf{e}_2$ is updated from $\mathbf{e}_1$ accordingly, or 2) there is a $\widetilde{\mathbf{x}}$ s.t. $\widetilde{\mathbf{x}}_1 \succ \widetilde{\mathbf{x}}$ and $\widetilde{\mathbf{x}} \succ \widetilde{\mathbf{x}}_2$.

Definition 1 (Redactable Signature Scheme [7]) *A redactable signature scheme with respect to binary relation $\vdash$, is a tuple of probabilistic polynomial time algorithms* $(\mathtt{Sign}, \mathtt{Verify}, \mathtt{Redact})$*, such that*

1. *for any message x, $\sigma = \mathtt{Sign}_{\mathcal{SK}}(x) \Rightarrow \mathtt{Verify}_{\mathcal{PK}}(x, \sigma) = \mathtt{TRUE}$;*
2. *for any messages x and y, suth that $x \vdash y$,*

$$\sigma = \mathtt{Sign}_{\mathcal{SK}}(x) \wedge \sigma' = \mathtt{Redact}_{\mathcal{PK}}(x, \sigma, y) \Rightarrow \mathtt{Verify}_{\mathcal{PK}}(y, \sigma') = \mathtt{TRUE},$$

where $\mathcal{PK}$ and $\mathcal{SK}$ are the public/private key respectively.

Both Johnson et al.[7] and Miyazaki et al.[11] presented a redactable signature scheme w.r.t superset relation. Johnson et al.[7] also gaves security definition for homomorphic signature schemes. We adapt their definition for redactable signature scheme. Let $\vdash$ denote a binary relation. For any set S, let $span_{\vdash}(S)$ denote the set $\{x : \exists y \in S, \text{ s.t. } y \vdash x\}$.

Definition 2 (Unforgeability of Redactable Signature Scheme [7]) *A redactable signature scheme $\langle \mathtt{Sign}, \mathtt{Verify}, \mathtt{Redact}\rangle$ is (t, q, ϵ) -unforgeable against existential forgeries with respect to $\vdash$, if any adversary $\mathcal{A}$ making at most q chosen-messge queries adaptively and running in time at most t, has advantage $Adv\mathcal{A} \le \epsilon$. The advantage of an adversary $\mathcal{A}$ is defined as the probability that, after queries on ℓ ($\ell \le q$) messages $x_1, x_2, \ldots, x_\ell$, $\mathcal{A}$ outputs a valid signature σ for some message $x \notin span_{\vdash}(\{x_1, x_2, \ldots, x_\ell\})$. Formally, let $(\mathcal{PK}, \mathcal{SK})$ be a public key pair,*

$$Adv\mathcal{A} = \mathsf{Pr}\left[\begin{array}{c}\mathcal{A}^{\mathtt{Sign}_{\mathcal{SK}}} = (x, \sigma) \wedge \mathtt{Verify}_{\mathcal{PK}}(x, \sigma) = \mathtt{TRUE} \\ \wedge\ x \notin span_{\vdash}(\{x_1, x_2, \ldots, x_\ell\})\end{array}\right].$$

Redactable signature schemes have an additional security requirement on privacy [1]: The adversary should not be able to derive any information about the removed substrings from a redacted string and its signature.

Definition 3 (Privacy Preserving) *A redactable signature scheme* $\langle \mathtt{Sign}, \mathtt{Verify}, \mathtt{Redact}\rangle$ *is privacy preserving if, for any annotated strings* $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}$, *such that* $\mathbf{x}_1 \succ \mathbf{x}$ *and* $\mathbf{x}_2 \succ \mathbf{x}$*, the following distributions* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are computationally indistinguishable:*

$$\mathcal{S}_1 = \{\sigma : \sigma = \mathtt{Redact}_{\mathcal{PK}}(\mathbf{x}_1, \mathtt{Sign}_{\mathcal{SK}}(\mathbf{x}_1; \mathbf{r}_1), \mathbf{x}; \mathbf{r}_2)\},$$
$$\mathcal{S}_2 = \{\sigma : \sigma = \mathtt{Redact}_{\mathcal{PK}}(\mathbf{x}_2, \mathtt{Sign}_{\mathcal{SK}}(\mathbf{x}_2; \mathbf{r}_1), \mathbf{x}; \mathbf{r}_2)\},$$

where $\mathbf{r}_1$ *and* $\mathbf{r}_2$ *are random coins used by* `Sign` *and* `Redact` *respectively, and public/private key* $(\mathcal{PK}, \mathcal{SK})$ *is randomly chosen.*

4 `S4`: Redactable Signature Scheme for Strings

We propose `S4`, a redactable signature scheme for strings that is able to hide the lengths of the removed substrings. Our approach is as follows: We first propose a hash function $\mathcal{H}$ that maps an annotated string $\widetilde{\mathbf{x}}$ and an auxiliary input $\mathbf{y}$ to an unordered set. This hash is "collision resistant" and satisfies some properties on substring removal. Using $\mathcal{H}$ and some known redactable signature schemes for unordered sets, we have a redactable signature scheme for strings.

4.1 Hashing strings to unordered sets

Let $\mathcal{H}$ be a hash function that maps an annotated string $\widetilde{\mathbf{x}}$ and an auxiliary input $\mathbf{y}$, which is a sequence of numbers from a finite field, to a (unordered) set of elements from some universe. In our construction (Table 1), $\mathcal{H}$ maps the input to a set of 3-tuples from $\mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z}_n$, where n is some chosen parameter.

Definition 4 (Collision Resistant) *We say that* $\mathcal{H}$ *is* (t, ϵ)*-collision-resistant if, for any algorithm* $\mathcal{A}$ *with running time at most* t*,*

$$\mathsf{Pr}\left[\widetilde{\mathbf{x}}_1 \not\succ \widetilde{\mathbf{x}}_2 \ \wedge \ \mathcal{H}(\widetilde{\mathbf{x}}_2, \mathbf{y}_2) \subset \mathcal{H}(\widetilde{\mathbf{x}}_1, \mathbf{y}_1)\right] \leq \epsilon,$$

where $(\widetilde{\mathbf{x}}_1, \widetilde{\mathbf{x}}_2, \mathbf{y}_2)$ *is the output of* $\mathcal{A}$ *on input* $\mathbf{y}_1$*, and the probability is taken over uniformly randomly chosen* $\mathbf{y}_1$ *and random coins used by* $\mathcal{A}$*.*

To be used in constructing a secure string redactable signature scheme, besides collision resistance, the hash function $\mathcal{H}$ is also required to be 1) "redactable": given $\widetilde{\mathbf{x}}_1$, $\widetilde{\mathbf{x}}_2$ and $\mathbf{y}_1$, such that $\widetilde{\mathbf{x}}_1 \succ \widetilde{\mathbf{x}}_2$, it is easy to find $\mathbf{y}_2$ such that $\mathcal{H}(\widetilde{\mathbf{x}}_1, \mathbf{y}_1) \supset \mathcal{H}(\widetilde{\mathbf{x}}_2, \mathbf{y}_2)$; 2) privacy preserving: $\mathcal{H}(\widetilde{\mathbf{x}}_2, \mathbf{y}_2)$ must not reveal any information about the removed substring. The property on privacy preserving is essential and used in the proof of Theorem 3. However, for simplicity, we will not explicitly formulate the requirement here.

4.2 Construction of $\mathcal{H}$

We present a hash function $\mathcal{H}(\cdot, \cdot)$ in Table 1 based on some hash function h.

Let n be a RSA modulus, and $h : \mathbb{Z}_n \to \mathbb{Z}_n$ be a hash function. Given $\mathbf{x} = x_1x_2 \ldots x_m$ associated with annotation $\mathbf{e}$, $\mathbf{r} = r_1r_2r_3 \ldots r_m$, and $\mathbf{w} = w_1w_2w_3 \ldots w_m$, where for each i, $x_i, r_i, w_i \in \mathbb{Z}_n$, i.e. $\mathbf{x}, \mathbf{r}$ and $\mathbf{w}$ are strings over alphabet $\mathbb{Z}_n$, we define $\mathcal{H}$ as

$$\mathcal{H}((\mathbf{x}, \mathbf{e}), (\mathbf{r}, \mathbf{w})) \triangleq \{t_i : t_i = (x_i,\ r_i,\ (w_i^{\prod_{j=1}^{i} h(r_j)} \mod n)), 1 \le i \le m\}.$$

Table 1. Definition of $\mathcal{H}(\cdot, \cdot)$.

Remarks on the construction of $\mathcal{H}$.

1. The witness $\mathbf{w}$ should be consistent with the annotation $\mathbf{e}$, because $\mathbf{w}$ is supposed to implicitly describe the locations of the removed substrings. For any $1 < i \le m$, $w_{i-1} \ne w_i$ means a non-empty substring has been removed just between x_{i-1} and x_i. If[2] $w_1 \ne g$, this indicates that a non-empty substring is removed from the front of the string $\mathbf{x}$. We assume that there are no substrings removed from the end of a string. Because of this correspondence, we may omit the explicit annotation $\mathbf{e}$ in the use of $\mathcal{H}$ for simplicity.
2. It is crucial that the value of r_i is explicitly represented in the t_i for each i (Table 1). If the r_i's are omitted, then it is easy to find collisions.

LEMMA 1 *The hash function $\mathcal{H}$ as defined in Table 1, is $(poly_1(k), \frac{1}{poly_2(k)})$-collision-resistant for any positive polynomials $poly_1(\cdot)$ and $poly_2(\cdot)$, where k is the security parameter, i.e. the bit length of n, assuming that h is division intractable[3] and always outputs odd prime integers, and Strong RSA Problem is hard.*

Basically, the proof reduces Strong RSA Problem or Division Problem [4] to the collision finding problem. Appendix A gives the proof sketch.

4.3 Construction of `S4`

We construct a redactable signature scheme `S4`, which consists of three algorithms `Sign`, `Verify`, and `Redact`, for strings with respect to binary relation $\succ$ based on the hash function $\mathcal{H}$ defined in Table 1 and some redactable signature scheme `SSS` for sets with respect to superset relation $\supseteq$.

The signer chooses a RSA modulus n and an element g of large order in $\mathbb{Z}_n^*$. Both n and g are public. Let the object space be $\mathbb{Z}_n$, that is, a string is a sequence of integers from $\mathbb{Z}_n$. Let $h : \mathbb{Z}_n \to \mathbb{Z}_n$ be a collision resistant hash which satisfies the division intractable property [4] and always outputs odd prime numbers[4].

[2] Here g is an element of large order from $\mathbb{Z}_n^*$. If there are no redaction at the front of $\mathbf{x}$, $w_1 = g$. See the use of $\mathcal{H}$ in Table 2.

[3] Division intractability [4] implies collision resistance.

[4] Gennaro [4] gave a way to construct hash function that is division intractable and always outputs odd prime numbers.

Let $\texttt{SSS} = (\texttt{sig}, \texttt{vrf}, \texttt{rec})$ be a redactable signature scheme for unordered sets w.r.t superset relation $\supseteq$. The signer also needs to choose the public and secret key pair $(\mathcal{PK}, \mathcal{SK})$, which is actually used by the underlying signature scheme SSS. The details of Sign, Verify, and Redact are presented in Table 2, 3 and 4.

The final signature of a string $x_1x_2\ldots x_m$ consists of m random numbers $r_1, r_2, \ldots, r_m$, the *witnesses* $w_1, w_2, \ldots, w_m$ where $r_i, w_i \in \mathbb{Z}_n$ for each i, and a signature $\mathbf{s}$ constructed by SSS. Recall that witness w_i's imply the annotation of string $x_1x_2\ldots x_m$.

Sign. Given $\mathbf{x} = x_1x_2\ldots x_m$ and its associated annotation $\mathbf{e} = \langle m \rangle$.

1. Choose a RSA modulus n, an element g of large order in $\mathbb{Z}_n$, and key $(\mathcal{PK}, \mathcal{SK})$. Make $n, g, \mathcal{PK}$ public.
2. Let $w_i = g$ for each i. Choose m distinct random numbers $r_1, r_2, \ldots, r_m$. Let $\mathbf{r} = r_1r_2r_3\ldots r_m$ and $\mathbf{w} = w_1w_2w_3\ldots w_m$. Compute
$$\mathbf{t} = \mathcal{H}((\mathbf{x}, \mathbf{e}), (\mathbf{r}, \mathbf{w})).$$
3. Sign the set $\mathbf{t}$ using SSS with the secret key $\mathcal{SK}$ to obtain $\mathbf{s}$:
$$\mathbf{s} = \texttt{sig}_{\mathcal{SK}}(\mathbf{t}).$$
4. The final signature consists of the random numbers r_i's, witnesses w_i's, and the signature $\mathbf{s}$. That is,
$$(\mathbf{r}, \mathbf{w}, \mathbf{s}) \text{ or } (r_1, r_2, \ldots, r_m; \quad w_1, w_2, \ldots, w_m; \quad \mathbf{s})$$

Table 2. S4: Sign.

Verify. Given a string $\mathbf{x} = x_1x_2\ldots x_m$ associated with annotation $\mathbf{e}$, its signature $(\mathbf{r}, \mathbf{w}, \mathbf{s})$, the public information n, g, and the public key $\mathcal{PK}$ of SSS.

1. If $\mathbf{e}$ and $\mathbf{w}$ are not consistent, output FALSE.
2. Compute $\mathbf{t} = \mathcal{H}(\mathbf{x}, (\mathbf{r}, \mathbf{w}))$.
3. $(\mathbf{r}, \mathbf{w}, \mathbf{s})$ is a valid signature of $\mathbf{x}$ under S4, if and only if $\mathbf{s}$ is a valid signature of $\mathbf{t}$ under SSS, i.e.
$$\texttt{vrf}_{\mathcal{PK}}(\mathbf{t}, \mathbf{s}) = \texttt{TRUE}.$$

Table 3. S4: Verify.

Theorem 2 S4 *is* $(t, q, \frac{\epsilon_1}{1-\epsilon_2})$*-unforgeable against existential forgeries with respect to relation* $\succ$*, if* SSS *is* $(t + qt_0, q, \epsilon_1)$*-unforgeable against existential forg-*

`Redact`. Given a string $\mathbf{x} = x_1x_2\dots x_m$ associated with annotation $\mathbf{e}$, and its signature $(\mathbf{r}, \mathbf{w}, \mathbf{s})$, where $\mathbf{r} = r_1r_2\dots r_m$, $\mathbf{w} = w_1w_2\dots w_m$, the public information n, g, public key $\mathcal{PK}$ for `SSS`, and (i, j) the location of the string to be removed (that is $x_ix_{i+1}\dots x_j$ is to be removed).

1. Update $\mathbf{e}$ to obtain new annotation $\hat{\mathbf{e}}$. Compute $u = \prod_{k=i}^{j} h(r_k)$, to update the witnesses in the following way: for each $\ell > j$, update w_ℓ
$$\hat{w}_\ell \leftarrow w_\ell^u \mod n.$$
2. Let $\hat{\mathbf{x}} = x_1x_2\dots x_{i-1}x_{j+1}\dots x_m$, $\hat{\mathbf{r}} = r_1r_2\dots r_{i-1}r_{j+1}\dots r_m$ and $\hat{\mathbf{w}} = w_1w_2\dots w_{i-1}\hat{w}_{j+1}\hat{w}_{j+2}\dots\hat{w}_m$. Compute
$$\hat{\mathbf{t}} = \mathcal{H}((\hat{\mathbf{x}}, \hat{\mathbf{e}}), (\hat{\mathbf{r}}, \hat{\mathbf{w}})).$$
3. Compute
$$\hat{\mathbf{s}} = \mathtt{rec}_{\mathcal{PK}}(\mathbf{t}, \mathbf{s}, \hat{\mathbf{t}})$$
where $\mathbf{t} = \mathcal{H}((\mathbf{x}, \mathbf{e}), (\mathbf{r}, \mathbf{w}))$.
4. Output $(\hat{\mathbf{r}}, \hat{\mathbf{w}}, \hat{\mathbf{s}})$ as the signature of $(\hat{\mathbf{x}}, \hat{\mathbf{e}})$.

Table 4. `S4`: `Redact`.

eries with respect to superset relation $\supseteq$, and $\mathcal{H}$ is $(t+qt_1, \epsilon_2)$-collision-resistant, where t_0 is the running time of $\mathcal{H}$ and t_1 is the time needed by `S4` *to sign a document.*

The proof is given in Appendix B. Our construction of $\mathcal{H}$ (Table 1) is collision resistant (Lemma 1). Johnson et al.[7] showed their redactable signature scheme `Sig` (in Section 5 of [7]) is (t, q, ϵ)-unforgeable under reasonable assumptions (see Theorem 1 in [7]), for some proper parameters t, q and ϵ. Miyazaki et al.[11] also showed a similar result on the unforgeability of the redactable signature scheme they proposed. Hence, conditions in Theorem 2 can be satisfied.

Theorem 3 *The redactable signature scheme* `S4` *is privacy preserving (as defined in Definition 3), assuming that hash function h satisfies the property: for any integer constants $a, b \geq 1$, the two distributions $\mathcal{X} = g^{h(U_1)h(U_2)\dots h(U_a)} \mod n$ and $\mathcal{Y} = g^{h(U'_1)h(U'_2)\dots h(U'_b)} \mod n$ are computational indistinguishable, where n is a RSA modulus, g is an element of large order in $\mathbb{Z}_n^*$ and U_i's and U'_j's are all independent uniform random variables over $\mathbb{Z}_n$.*

If $h(U)$ is indistinguishable from the uniform random input U, then h satisfies the property in Theorem 3.

4.4 Efficiency

The size of $\mathbf{s}$ depends on `SSS`, and let us assume it requires κ bits. The number of distinct w_i's is about the same as the number of redactions occurred. So w_i's can be represented in $t(k + \lceil \log m \rceil)$ bits, where t is the number of substrings removed, and k is the bit length of n. Thus the total number of bits required is at most $k(m + t) + t\lceil \log m \rceil + \kappa$. One may ignore the term $t\lceil \log m \rceil$ since it is used to specify the locations of removed substrings, which could be treated as part of the message. Hence, the size of signature is essentially $km + kt + \kappa$. The dominant term is km, which is the total size of the random numbers r_i's.

Disregarding the time taken by the scheme SSS, and the time required to compute the hash $h(\cdot)$, during signing, $O(m)$ of k-bits exponentiation operations are required. During redaction, if ℓ consecutive objects are to be removed between position i and j, and t' number of redactions have been made after position j, then the number of k-bit exponentiation operations is at most $\ell(t' + 1)$, which is in $O(\ell m)$. During verification, $O(tm)$ number of k-bits exponentiation operations are required. Hence, our scheme is suitable for small t, which is reasonable in practice. In sum, the main drawback of S4 is the size of its signature. In the next section, we will reduce its size using a random tree.

5 RGGM: Random tree with Hiding property

We propose RGGM, a variant of GGM tree [5] to generate a sequence of pseudo random numbers, where the structure of the tree is randomized. This generator provides us with the ability to remove multiple substrings[5] of pseudo random numbers, while still being able to generate the retained numbers from a short seed. The expected size of the new seed is in $O(k + tk \log m)$ where t is the number of removed substrings, m is the number of pseudo random numbers, and k is a security parameter. More importantly, the new seed does not reveal any information about the size nor the content of the removed substrings.

Pseudo random number generation. To generate m pseudo random numbers we employ a method similar to that in the redactable signature scheme proposed by Johnson et al. [7], which is based on the GGM tree [5]. Let $G : \mathcal{K} \rightarrow \mathcal{K} \times \mathcal{K}$ be a length-doubling pseudo random number generator. First pick an arbitrary binary tree T with m leaves, where all internal nodes of T have exactly two children, the left and right child. Next, pick a seed $t \in \mathcal{K}$ uniformly at random, and associate it with the root. The pseudo random numbers $r_1, r_2, \ldots, r_m$ are then computed from t in the usual top-down manner along the binary tree.

Hiding random numbers. If r_i is to be removed, the associated leaf node and all its ancestors will be removed, as illustrated by the example in Figure 1(b). The values associated with the roots of the remaining subtrees, and a description of the structure of the subtrees, form the new seed, whereby the remaining random values r_j's ($j \neq i$) can be re-computed. By the property of G, it is computationally difficult to guess the removed value r_i from the new seed.

[5] A substring is as a subsequence where all the elements are consecutive elements in the full sequence.

```
TreeGen: Given $m$, output a binary tree $T$ with $m$ leaves:

1. Pick a $p$ uniformly at random from $\{1, 2, \ldots, m-1\}$.
2. Recursively generate a tree $T_1$ with $p$ leaves.
3. Recursively generate a tree $T_2$ with $m-p$ leaves.
4. Output a binary tree with $T_1$ as the left subtree and $T_2$ as the right subtree.
```

Table 5. `TreeGen`: a random tree generation algorithm

Unlike the method proposed by Johnson et al. [7], our tree T is randomly generated. If the tree is known to be balanced (or known to be of some fixed structure), some information on the number of leaf nodes removed can be derived from the redacted tree. Our random trees are generated by the probabilistic algorithm `TreeGen` in Table 5. Note that descriptions of the structure of the tree are required for the regeneration of the random values r_i's.

At the moment, for ease of presentation, the descriptions are stored together with the seed. This increases the size of the seed. To reduce the size, we can replace the description by another short random seed $\hat{t}$, which is assigned to the root. The random input required in Step 1 of the algorithm can be generated from $\hat{t}$ using G. A difference between the two methods of storing the (redacted) tree structure information is that in the former, we will have a information theoretic security result, whereas in the later, the security depends on the security of G.

Our *main observation* is as follows: after a substring of leaves is removed from the random tree, the remaining subtrees do not reveal (information theoretically) anything about the number of leaves removed, except the fact that at least one leaf has been removed at that location.

Notations. Given a binary tree T, its leaf nodes can be listed from the left to right to obtain a sequence. We call a subsequence of consecutive leaves a substring of leaves. After multiple substrings of leaves and all of their ancestor nodes are deleted, the remaining structures form a *redacted* tree[6] represented by two sequences, $\mathbf{T} = \langle T_1, T_2, \ldots, T_v \rangle$ and $\mathbf{b} = \langle m, b_1, b_2, \ldots, b_u \rangle$. where T_i's are the subtrees retained, and each b_i indicates that a substring was removed between the b_i-th and $(b_i + 1)$-th locations in the remaining sequence of leaf nodes. Let q_i be the number of leaves that were removed in this substring. We call the sequence $\langle m, (b_1, q_1), (b_2, q_2), \ldots, (b_u, q_u) \rangle$ the *original annotation* of $\mathbf{b}$. Thus, the total number of leaf nodes removed is $\sum_{i=1}^{u} q_i$.

Let us consider this process. Given an original annotation $\mathbf{b}_1$, for a string of size m, a random tree T of size m is generated using `TreeGen`, and then redacted according to $\mathbf{b}_1$. Let $\texttt{RED}(\mathbf{b}_1)$ be the redacted tree.

From an adversary's point of view, he has $\texttt{RED}(\mathbf{b}_1)$, represented as $(\mathbf{T}, \mathbf{b})$, and wants to guess the q_i's in the original annotation $\mathbf{b}_1$, or the original total

[6] Although strictly speaking it is a forest.

number of leaf nodes. We want to show that the additional knowledge of $\mathbf{T}$ does not improve his chances, compared to another adversary who only has $\mathbf{b}$. It is suffice to show that, given any $\mathbf{b}$ and any two possible original annotations $\mathbf{b}_1$ and $\mathbf{b}_2$, the conditional probabilities of obtaining $(\mathbf{T}, \mathbf{b})$ are the same. That is,

Theorem 4 *For any redacted tree* $(\mathbf{T}, \mathbf{b})$*, any distribution* $\mathcal{B}$ *on the original annotation, and any* $\mathbf{b}_1$*,* $\mathbf{b}_2$*,*

$$Prob(\texttt{RED}(\mathcal{B}) = (\mathbf{T}, \mathbf{b}) \mid \mathcal{B} = \mathbf{b}_1) = Prob(\texttt{RED}(\mathcal{B}) = (\mathbf{T}, \mathbf{b}) \mid \mathcal{B} = \mathbf{b}_2)$$

6 S5: A Short Redactable Signature Scheme for Strings

S4 produces a large signature, whose main portion is a sequence of true random numbers r_i's. We can combine RGGM with S4 to produce a short signature by replacing the r_i's with pseudo random numbers generated by RGGM. Let us call this combined scheme S5, short redactable signature scheme for strings.

It is easy to show S5 is secure and privacy preserving from Theorem 2, Theorem 3, Lemma 1 and the fact that RGGM is a pseudo random number generator. Due to space constraint, we give the relevant statements in Appendix E.

Efficiency. The improvement of S5 is in signature size. Given the unredacted string, the size of the signature is $\kappa + 2k$, where κ is the signature size of SSS, and k is the length of each seed. Recall that we need two seeds in RGGM, one for the generation of the numbers, and the other for the tree structure.

If t substrings are removed, the signature size is $\kappa + tk + O(kt \log m)$, where the term tk is for the witness, and $O(kt \log m)$ is required for the RGGM.

7 Conclusion

We considered a simple but difficult requirement in redactable signature scheme: hiding the lengths of the removed substrings. We exploited an intriguing statistical property of random trees, and employed a hash from strings to unordered sets to achieve the requirement. Although the signature is short, its size still depends on the number of substrings removed and the length of the string. In contrast, there are known schemes for unordered sets, whose signature size is a constant. Hence, it is interesting to find out whether it is possible to close the gap. The computation time required during verification is high for long strings when many substrings are removed. It is also interesting to find a practical way to organize the string hierarchically, so as to achieve speedup.

The two main components, the hash $\mathcal{H}$ and the RGGM tree, proposed in this paper, could be of independent interests. The hash function may play a role in the design of transitive signature with additional property on privacy preservation. Many secure outsourced database applications involve Merkel tree or GGM tree. The hiding property of the RGGM tree may be useful in those applications.

References

1. Giuseppe Ateniese, Daniel Chou, Breno Medeiros, and Gene Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177, 2005.
2. Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *CRYPTO*, pages 216–233, 1994.
3. Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In *STOC*, pages 45–56, 1995.
4. Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In *EUROCRYPT*, pages 123++, 1999.
5. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
6. Tetsuya Izu, Nobuyuki Kanaya, Masahiko Takenaka, and Takashi Yoshioka. Piats: A partially sanitizable signature scheme. In *ICICS*, pages 72–83, 2005.
7. Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262, 2002.
8. Marek Klonowski and Anna Lauks. Extended sanitizable signatures. In *ICISC*, pages 343–355, 2006.
9. Ralph Merkle. Protocols for public key cryptosystems. In *SP*, page 122, 1980.
10. Silvio Micali and Ronald Rivest. Transitive signature schemes. In *CT-RSA*, pages 236–243, 2002.
11. Kunihiko Miyazaki, Goichiro Hanaoka, and Hideki Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *ASIACCS*, pages 343–354, 2006.
12. Ronald Rivest. Two new signature schemes. Presented at Cambridge seminar, 2001. http://www.cl.cam.ac.uk/Research/Security/seminars/2000/rivest-tss.pdf.
13. Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In *ICISC*, pages 163–205, 2001.
14. Manabu Suzuki, Isshiki Toshiyuki, and Keisuke Tanaka. Sanitizable signature with secret information. Technical report, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2006.
15. Xun Yi. Directed transitive signature scheme. In *CT-RSA*, pages 129–144, 2007.

A Proof of Lemma 1

LEMMA 1. The hash function $\mathcal{H}$ as defined in Table 1, is $(poly_1(k), \frac{1}{poly_2(k)})$-collision-resistant for any positive polynomials $poly_1(\cdot)$ and $poly_2(\cdot)$, where k is the security parameter, i.e. the bit length of n, assuming that h is division intractable[7] and always outputs odd prime integers, and Strong RSA Problem is hard.

Proof. (Sketch) Suppose there exists a probabilistic polynomial time algorithm $\mathcal{A}$, which takes as input $(\mathbf{r}_1, \mathbf{w}_1)$, and outputs $(\mathbf{x}_1, \mathbf{e}_1), (\mathbf{x}_2, \mathbf{e}_2)$, and $(\mathbf{r}_2, \mathbf{w}_2)$, such that $((\mathbf{x}_2, \mathbf{e}_2), (\mathbf{r}_2, \mathbf{w}_2))$ is a collision with $((\mathbf{x}_1, \mathbf{e}_1), (\mathbf{r}_1, \mathbf{w}_1))$ with noticeable probability.

[7] Division intractability [4] implies collision resistance.

We first consider the special case where $\mathbf{w}_1 = \underbrace{\langle g, g, \ldots, g\rangle}_{m\ g\text{'s}}$ and $\mathbf{e}_1 = \langle m\rangle$. Let $\mathbf{x}_1 = x_{1,1}x_{1,2}\ldots x_{1,m}, \mathbf{x}_2 = x_{2,1}x_{2,2}\ldots x_{2,m'}, \mathbf{r}_1 = \langle r_{1,1}, r_{1,2}, \ldots, r_{1,m}\rangle, \mathbf{r}_2 = \langle r_{2,1}, r_{2,2}, \ldots, r_{2,m'}\rangle, \mathbf{w}_2 = \langle w_1, w_2, \ldots, w_{m'}\rangle$ for some $m' < m$. By the definition of collision, we have

$$\begin{gathered}(\mathbf{x}_1, \mathbf{e}_1) \not\succ (\mathbf{x}_2, \mathbf{e}_2) \quad \wedge \\ \mathcal{H}((\mathbf{x}_2, \mathbf{e}_2), (\mathbf{r}_2, \mathbf{w}_2)) = \{t_{2,1}, \ldots, t_{2,m'}\} \subset \\ \mathcal{H}((\mathbf{x}_1, \mathbf{e}_1), (\mathbf{r}_1, \mathbf{w}_1)) = \{t_{1,1}, \ldots, t_{1,m}\},\end{gathered}$$

where for each i, $t_{1,i} = (x_{1,i},\ r_{1,i},\ (g^{\prod_{j=1}^{i} h(r_{1,j})} \mod n))$ and $t_{2,i} = (x_{2,i},\ r_{2,i},\ (w_i{}^{\prod_{j=1}^{i} h(r_{2,j})} \mod n))$.

There are two different cases of collisions: 1) $\mathbf{x}_2$ is not a substring of $\mathbf{x}_1$, e.g. $\mathbf{x}_1 = abc, \mathbf{x}_2 = cb$; 2) $\mathbf{x}_2$ is a substring of $\mathbf{x}_1$ but their annotations $\mathbf{e}_1$ and $\mathbf{e}_2$ are incompatible, e.g. $\mathbf{x}_1 = abc$, $\mathbf{w}_1 = ggg, \mathbf{e}_1 = \langle 3\rangle$, and $\mathbf{x}_2 = ac$, $\mathbf{w}_2 = gg, \mathbf{e}_2 = \langle 2\rangle$.

Case 1: $\mathbf{x}_2$ is not a substring of $\mathbf{x}_1$. the sequence $\langle t_{2,1}, t_{2,2}, \cdots, t_{2,m'}\rangle$ must be a repermutation of some subsequence of the sequence $\langle t_{1,1}, t_{1,2}, \cdots, t_{1,m}\rangle$. Thus, there must be a swap of positions, i.e. there are indices a, b, c and d such that $a < b$, $c < d$, and $t_{1,a} = t_{2,d}$ and $t_{1,b} = t_{2,c}$. That means,

$$(x_{1,a},\ r_{1,a},\ g^{A}) = (x_{2,d},\ r_{2,d},\ w_d^{EF}), \text{ and } (x_{1,b},\ r_{1,b},\ g^{AB}) = (x_{2,c},\ r_{2,c},\ w_c^{E}),$$

where $A = \prod_{i=1}^{a} h(r_{1,i}), B = \prod_{i=a+1}^{b} h(r_{1,i}), E = \prod_{i=1}^{c} h(r_{2,i}), F = \prod_{i=c+1}^{d} h(r_{2,i})$.

Next we can conduct a case analysis based on whether EF divides A, to find a contradiction with our assumptions. If EF does not divide A (with noticeable probability), we could solve the Strong RSA Problem based on algorithm $\mathcal{A}$ using similar techniques in the proof of Theorem 5 in [4]; otherwise we could solve the Division Problem [4].

Case 2: $\mathbf{x}_2$ is a substring of $\mathbf{x}_1$ but they have incompatible annotations. Let $\mathbf{e}_3$ be the annotation such that $(\mathbf{x}_1, \mathbf{e}_1) \succ (\mathbf{x}_2, \mathbf{e}_3)$. Let $\mathbf{w}_3 = w_{3,1}\ldots w_{3,m'}$ be the witness obtained from step 1 of Table 4 when redacting $((\mathbf{x}_1, \mathbf{e}_1), (\mathbf{r}_1, \mathbf{w}_1))$ to produce a signature for the annotated substring $(\mathbf{x}_2, \mathbf{e}_3)$. We know

$$\mathcal{H}((\mathbf{x}_2, \mathbf{e}_3), (\mathbf{r}_2, \mathbf{w}_3)) \subset \mathcal{H}((\mathbf{x}_1, \mathbf{e}_1), (\mathbf{r}_1, \mathbf{w}_1))$$

and

$$\mathcal{H}((\mathbf{x}_2, \mathbf{e}_2), (\mathbf{r}_2, \mathbf{w}_2)) \subset \mathcal{H}((\mathbf{x}_1, \mathbf{e}_1), (\mathbf{r}_1, \mathbf{w}_1)).$$

Because distinct random numbers $r_{1,i}$'s and $r_{2,j}$'s are involved in the output of $\mathcal{H}$, we conclude that $\mathcal{H}((\mathbf{x}_2, \mathbf{e}_3), (\mathbf{r}_2, \mathbf{w}_3)) = \mathcal{H}((\mathbf{x}_2, \mathbf{e}_2), (\mathbf{r}_2, \mathbf{w}_2))$. We also know $\mathbf{w_2}$ and $\mathbf{w_3}$ implies different annotations $\mathbf{e}_2$ and $\mathbf{e}_3$, so $\mathbf{w}_2 \neq \mathbf{w}_3$. That means there exists an index $i, 1 \leq i \leq m'$, such that

$$w_{2,i} \neq w_{3,i} \text{ and } w_{2,i}^{Q} = w_{3,i}^{Q} \mod n, \tag{1}$$

where $Q = \prod_{j=1}^{i} h(r_{2,j})$ is an odd number. Recall that $n = pq$, $p = 2p' + 1, q = 2q' + 1$ and p, q, p', q' are all prime numbers. Let $\lambda(n) = lcm(p-1, q-1) = 2p'q'$. Suppose Q is coprime with $\lambda(n)$. Then $w_{2,i}^Q = w_{3,i}^Q \mod n$ implies $w_{2,i} = w_{3,i}$, which is a contradiction with Eq 1. So we conclude that the odd number Q and $\lambda(n) = 2p'q'$ have common factors, i.e., p' or q' divides Q. W.L.O.G., we assume $p'|Q$.

Because $h(\cdot)$ outputs odd prime number only, there must exist $\xi, 1 \le \xi \le m'$, $h(r_{2,\xi}) = p'$. Consequently, $2h(r_{2,\xi}) + 1$ divides n, and we can factorize n efficiently by trying every $h(r_{2,i}), 1 \le i \le m'$.

In the beginning of the proof, we assume that $\mathbf{w}_1 = \langle g, \dots, g \rangle$. For case 1, it is easy to generalize the proof for general $\mathbf{w}_1 = \langle w_{1,1}, w_{1,2}, \dots, w_{1,m} \rangle$ using similar techniques in the proof of Theorem 5 in [4], by randomly guessing the index a and setting $w_{1,a} = s$; for case 2, there is no need of changes in the proof.

B Proof of Theorem 2

Theorem 2. The scheme $\mathtt{S4}$ as constructed in Table 2, Table 3 and Table 4, is $(t, q, \frac{\epsilon_1}{1-\epsilon_2})$-secure against existential forgeries with respect to $\succ$, if $\mathtt{SSS}$ is $(t + qt_0, q, \epsilon_1)$-secure against existential forgeries with respect to $\supseteq$, and $\mathcal{H}$ is $(t + qt_1, \epsilon_2)$-collision-resistant, where t_0 is the running time of $\mathcal{H}$ and t_1 is the time needed by $\mathtt{S4}$ to sign a document.

Proof: We prove this theorem using proof by contradiction. Let $\mathcal{O}^{\mathtt{S4}}$ and $\mathcal{O}^{\mathtt{SSS}}$ denote the oracles for Sign algorithms of $\mathtt{S4}$ and $\mathtt{SSS}$ respectively, and both use the same public/private key $(\mathcal{PK}, \mathcal{SK})$. Suppose there exists an adversary $\mathcal{A}$ against $\mathtt{S4}$, such that $\mathcal{A}$ runs in time at most t, makes at most q chosen-message queries to $\mathcal{O}^{S4}$, and has advantage $Adv\mathcal{A} > \epsilon_3 = \frac{\epsilon_1}{1-\epsilon_2}$. Let $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_\ell$ denote the $\ell (\le q)$ messages (annotated strings of objects) chosen by $\mathcal{A}$, and $(\sigma_i, \mathbf{r}_i)$ be the reply from $\mathcal{O}^{S4}$ for query $\mathbf{x}_i$. Let $(\mathbf{x}, (\sigma, \mathbf{r}))$ denote the output of $\mathcal{A}$.

$$Adv\mathcal{A} = \mathsf{Pr}\left(\mathtt{Verify}_{\mathcal{PK}}(\mathbf{x}, (\mathbf{r}, \sigma)) = \mathtt{TRUE} \ \wedge \ \mathbf{x} \notin span_{\succ}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_\ell)\right) > \epsilon_3.$$

Now we construct an adversary $\mathcal{B}$ against $\mathtt{SSS}$ invoking $\mathcal{A}$ as a subroutine and simulating $\mathcal{O}^{\mathtt{S4}}$ with $\mathcal{O}^{\mathtt{SSS}}$:

1. Run algorithm $\mathcal{A}$.
2. Receive query $\mathbf{x}_i$ from $\mathcal{A}$.
3. Choose a random coin $\mathbf{r}_i$, and compute $\mathcal{H}(\mathbf{x}_i, \mathbf{r}_i)$. Make query $\mathcal{H}(\mathbf{x}_i, \mathbf{r}_i)$ to $\mathcal{O}^{\mathtt{SSS}}$ and get reply σ_i.
4. Reply $(\sigma_i, \mathbf{r}_i)$ to $\mathcal{A}$ corresponding to the query $\mathbf{x}_i$.
5. If receving another qeury from $\mathcal{A}$, go to Step 2.
6. Receive output $(\mathbf{x}, (\sigma, \mathbf{r}))$ from $\mathcal{A}$.
7. Output $(\mathcal{H}(\mathbf{x}, \mathbf{r}), \sigma)$.

The number of queries made to $\mathcal{O}^{\mathtt{SSS}}$ by $\mathcal{B}$ equals to the number of queries made to (simulated) $\mathcal{O}^{\mathtt{S4}}$ by $\mathcal{A}$. The running time of $\mathcal{B}$ is at most $t + qt_0$.

With probability larger than ϵ_3, $(\mathbf{x}, (\sigma, \mathbf{r}))$ is an existential forgery w.r.t to S4, formally

$$\mathsf{Pr}\left(\mathtt{Verify}_{\mathcal{PK}}(\mathbf{x}, (\mathbf{r}, \sigma)) = \mathtt{TRUE} \ \wedge \ \mathbf{x} \notin \mathit{span}_{\succ}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell)\right) > \epsilon_3.$$

Suppose $\mathbf{x}_i \not\succ \mathbf{x}$ for any i. Because $\mathcal{H}$ is $(t+qt_1, \epsilon_2)$-collision-resistant and real running time (taking the time for signing into account) of $\mathcal{A}$ is at most $t + qt_1$, we have

$$\mathsf{Pr}\left(\vee_{i=1}^{\ell}\left(\mathcal{H}(\mathbf{x}, \mathbf{r}) \subseteq \mathcal{H}(\mathbf{x}_i, \mathbf{r}_i)\right)\right) \leq \epsilon_2.$$

Hence,

$$\mathsf{Pr}\begin{pmatrix} \mathtt{vrf}_{\mathcal{PK}}(\mathcal{H}(\mathbf{x}, \mathbf{r}), \sigma) = \mathtt{TRUE} \ \wedge \\ \mathcal{H}(\mathbf{x}, \mathbf{r}) \notin \mathit{span}_{\supseteq}(\mathcal{H}(\mathbf{x}_1, \mathbf{r}_1), \mathcal{H}(\mathbf{x}_2, \mathbf{r}_2), \ldots, \mathcal{H}(\mathbf{x}_\ell, \mathbf{r}_\ell)) \end{pmatrix} > \epsilon_3(1 - \epsilon_2) = \epsilon_1.$$

That means $\mathcal{B}$ is an existential forger against SSS: $\mathcal{B}$ runs in time at most $t + qt_0$, makes at most q queries to $\mathcal{O}^{\mathtt{SSS}}$, and has advantage larger than ϵ_1. This is a contradiction with our assumption that SSS is $(t + qt_0, q, \epsilon_1)$-secure against existential forgeries.

Hence, our hypothesis that there exists such adversary $\mathcal{A}$ is wrong. In other words, S4 is $(t, q, \frac{\epsilon_1}{1-\epsilon_2})$-secure against existential forgeries with respect to $\succ$. $\square$

C Proof of Theorem 3

THEOREM 3. The redactable signature scheme S4 is privacy preserving (as defined in Definition 3), assuming that hash function h satisfies the property: for any integer constants $a, b \geq 1$, the two distributions $\mathcal{X} = g^{h(U_1)h(U_2)\ldots h(U_a)}$ mod n and $\mathcal{Y} = g^{h(U_1')h(U_2')\ldots h(U_b')}$ mod n are computational indistinguishable, where n is a RSA modulus, g is an element of large order in $\mathbb{Z}_n^*$ and U_i's and U_j''s are all independent uniform random variables from $\mathbb{Z}_n$.

Proof: W.L.O.G, we represent the three strings $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}$ as stated in Definition 3 in following way:

$$\begin{aligned}
\mathbf{x}_1 &= x_1x_2x_3\ldots x_{a_1}\mathbf{y}_1x_{a_1+1}\ldots x_{a_2}\mathbf{y}_2x_{a_2+1}\ldots x_{a_\ell}\mathbf{y}_\ell x_{a_\ell+1}\ldots x_m \\
\mathbf{x}_2 &= x_1x_2x_3\ldots x_{a_1}\mathbf{z}_1x_{a_1+1}\ldots x_{a_2}\mathbf{z}_2x_{a_2+1}\ldots x_{a_\ell}\mathbf{z}_\ell x_{a_\ell+1}\ldots x_m \\
\mathbf{x} &= x_1x_2x_3\ldots x_{a_1}\diamond x_{a_1+1}\ldots x_{a_2}\diamond x_{a_2+1}\ldots x_{a_\ell}\diamond x_{a_\ell+1}\ldots x_m
\end{aligned}$$

$\mathbf{x}$ is a string of size m with ℓ substrings being redacted at locations $a_1, a_2, a_3, \ldots, a_\ell$. $\mathbf{x}$ can be obtained from $\mathbf{x}_1$ by removing non-empty substrings $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_\ell$, or from $\mathbf{x}_2$ by removing non-empty substrings $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_\ell$.

For any sequence $\mathbf{b} = \langle b_1, b_2, \ldots, b_m\rangle$, define $h(\mathbf{b})$ as

$$h(\mathbf{b}) = \prod_{i=1}^{m} h(b_i).$$

Denote with $\mathbf{r}_{y,i}$ (or $\mathbf{r}_{z,i}$) the sequence of random numbers associated with substring $\mathbf{y}_i$ (or $\mathbf{z}_i$) respectively. Denote with $\mathbf{r}_1 = \langle r_{1,1}, r_{1,2}, \ldots, r_{1,m}\rangle$ the sequence of random numbers, which are associated with x_i's and chosen when signing $\mathbf{x}_1$. Denote with $\mathbf{r}_2 = \langle r_{2,1}, r_{2,2}, \ldots, r_{2,m}\rangle$ the sequence of random numbers, which are associated with x_i's and chosen when signing $\mathbf{x}_2$. All $r_{1,i}$ and $r_{2,i}$ are independent uniform random numbers over the set of odd numbers in $\mathbb{Z}_n$.

The signature of $\mathbf{x}$ under S4 by redacting $\mathbf{x}_1$ is

$$\begin{aligned}\sigma_1 &= \texttt{Redact}_{\mathcal{PK}}(\mathbf{x}_1, \texttt{Sign}_{\mathcal{SK}}(\mathbf{x}_1), \mathbf{x}) \\ &= (\delta_1, \mathbf{r}_1 = \langle r_{1,1}, r_{1,2}, r_{1,3}, \ldots, r_{1,m}\rangle, \mathbf{w}_1 = \langle w_{1,1}, w_{1,2}, w_{1,3}, \ldots, w_{1,m}\rangle)\end{aligned}$$

where for each $0 \le i \le \ell$, assuming $a_0 = 0, a_{\ell+1} = m+1, h(\mathbf{r}_{y,0}) = 1$,

$$w_{1,a_i+1} = w_{1,a_i+2} = \ldots = w_{1,a_{i+1}} = g^{\prod_{j=0}^{i} h(\mathbf{r}_{y,j})} \mod n \tag{2}$$

and δ_1 is the signature under SSS of the following set H_1

$$H_1 = \mathcal{H}(\mathbf{x}, (\mathbf{r}_1, \mathbf{w}_1)) = \left\{ s : s = \left(x_i, r_{1,i}, \left(w_{1,i}^{\prod_{j=1}^{i} h(r_{1,j})} \mod n \right) \right) \right\}.$$

Similarly, the signature of $\mathbf{x}$ under S4 by redacting $\mathbf{x}_2$ is

$$\begin{aligned}\sigma_2 &= \texttt{Redact}_{\mathcal{PK}}(\mathbf{x}_2, \texttt{Sign}_{\mathcal{SK}}(\mathbf{x}_2), \mathbf{x}) \\ &= (\delta_2, \mathbf{r}_2 = \langle r_{2,1}, r_{2,2}, r_{2,3}, \ldots, r_{2,m}\rangle, \mathbf{w}_2 = \langle w_{2,1}, w_{2,2}, w_{2,3}, \ldots, w_{2,m}\rangle)\end{aligned}$$

where for each $0 \le i \le \ell$, assuming $a_0 = 0, a_{\ell+1} = m+1, h(\mathbf{r}_{z,0}) = 1$,

$$w_{2,a_i+1} = w_{2,a_i+2} = \ldots = w_{2,a_{i+1}} = g^{\prod_{j=0}^{i} h(\mathbf{r}_{z,j})} \mod n \tag{3}$$

and δ_2 is the signature under SSS of the following set H_2

$$H_2 = \mathcal{H}(\mathbf{x}, (\mathbf{r}_2, \mathbf{w}_2)) = \left\{ s : s = \left(x_i, r_{2,i}, \left(w_{2,i}^{\prod_{j=1}^{i} h(r_{2,j})} \mod n \right) \right) \right\}.$$

Note $w_{1,i}$'s and $w_{2,i}$'s can be written in form of $g^{\prod h(\cdot)} \mod n$ as in Equation 2 and 3. So $\mathbf{w}_1$ and $\mathbf{w}_2$ are indistinguishable. Furthermore, $w_{1,i}^{\prod_{j=1}^{i} h(r_{1,j})} \mod n$ and $w_{2,i}^{\prod_{j=1}^{i} h(r_{2,j})} \mod n$ are indistinguishable for eachi i. As a result H_1 is indistinguishable from H_2, and δ_1 is indistinguishable from δ_2.

Hence, σ_1 and σ_2 are computationally indistinguishable. □

D Proof of Theorem 4

THEOREM 4. For any redacted tree $(\mathbf{T}, \mathbf{b})$, any distribution $\mathcal{B}$ on the original annotation, and any $\mathbf{b}_1$, $\mathbf{b}_2$,

$$\text{Prob}(\texttt{RED}(\mathcal{B}) = (\mathbf{T}, \mathbf{b}) \mid \mathcal{B} = \mathbf{b}_1) = \text{Prob}(\texttt{RED}(\mathcal{B}) = (\mathbf{T}, \mathbf{b}) \mid \mathcal{B} = \mathbf{b}_2)$$

Proof:

1. Consider the random tree generation algorithm `TreePermGen` in Table 6, which takes in random permutations as the source of randomness. The distribution of the random trees generated by `TreePermGen` is the same as the distribution by `TreeGen` (Table 5). To see that, consider the root and its left subtree. The distribution of the size of the left subtree is the same for both algorithms. This observation can be generalized to the distribution of the whole tree.

`TreePermGen`: Given m, output a binary tree T with m leaves. During the execution of this algorithm, each node in T is associated with a subset of $\{1, 2, \ldots, m\}$. Denote by π_m a permutation on m elements $\{1, 2, \ldots, m\}$.

```
1. Randomly pick a $\pi_{m-1}$.
2. Let $\mathcal{S}$ be a collection of sets, which is initially empty. Let $T$ to be an empty tree.
3. Let $A = \{1, 2, 3, \ldots, m\}$ and associate it to a new node $p$. Add $A$ into $\mathcal{S}$, and insert
   $p$ into $T$ as the root node.
4. For $i = 1$ to $m - 1$,
   (a) Find the set $A \in \mathcal{S}$ s.t. $\pi_{m-1}(i) \in A$. Let $p$ be the node that is associated with
       $A$. Remove $A$ from $\mathcal{S}$.
   (b) Divide $A$ into two sets $A_{\texttt{left}}$ and $A_{\texttt{right}}$, where
           $A_{\texttt{left}} = A \cap \{1, 2, \ldots, \pi_{m-1}(i)\}$,   and
           $A_{\texttt{right}} = A \cap \{\pi_{m-1}(i) + 1, \ldots, m\}$.
   (c) Create two children for $p$ and insert them into $T$. Associate $A_{\texttt{left}}$ and $A_{\texttt{right}}$
       to the left and right child respectively. Add $A_{\texttt{left}}$ and $A_{\texttt{right}}$ into $\mathcal{S}$.
5. Output the final tree $T$, without the sets associated to its nodes.
```

Table 6. `TreePermGen`, a tree generation algorithm using random permutation.

Given a permutation π and an original annotation $\mathbf{b}'$, let us write $\mathcal{A}(\pi)$ as the tree generated by `TreePermGen`. Recall that the tree is to be redacted by removing some leaves and all their ancestors, and the leaves to be removed are specified by an original annotation $\mathbf{b}'$. Let us write $\mathcal{A}(\pi, \mathbf{b}')$ as the redacted tree.

2. We now analyze original annotations of these special forms:

$$\mathbf{b}_1 = \langle m, (b_1, q_1), (b_1, q_2), \ldots, (b_{i_0-1}, q_{i_0-1}), (b_{i_0}, k), (b_{i_0+1}, q_{i+1}) \ldots, (b_u, q_u)\rangle$$
$$\mathbf{b}_2 = \langle m+1, (b_1, q_1), (b_1, q_2), \ldots, (b_{i_0-1}, q_{i_0-1}), (b_{i_0}, k+1), (b_{i_0+1}+1, q_{i_0+1}) \ldots, (b_u+1, q_u)\rangle$$

for some i_0 and u. Under annotation $\mathbf{b}_1$, the i_0-th substring that is being removed has k items. Let the last item in this i_0-th removed substring be the v-th item in the original string, that is, $v = k + b_{i_0}$. In contrast, under annotation $\mathbf{b}_2$ the i_0-th substring that is removed has $k + 1$ items. Note that the tree on which $\mathbf{b}_1$

is applied has m leaf nodes, and the tree on which $\mathbf{b}_2$ is applied has $m+1$ leaf nodes.

We want to show that, for any redacted tree, represented as $(\mathbf{T}, \mathbf{b})$, and any $\mathbf{b}_1$ and $\mathbf{b}_2$ as defined above, we have

$$m\,|\{\pi_{m-1} : \mathcal{A}(\pi_{m-1}, \mathbf{b}_1) = (\mathbf{T}, \mathbf{b})\}| \;=\; |\{\pi_m : \mathcal{A}(\pi_m, \mathbf{b}_2) = (\mathbf{T}, \mathbf{b})\}| \tag{4}$$

Since a permutation is uniformly chosen in Step 1 of `TreePermGen`, the conditional probability of obtaining $(\mathbf{T}, \mathbf{b})$ given $\mathbf{b}_1$ is

$$\frac{1}{(m-1)!}\,|\{\pi_{m-1} : \mathcal{A}(\pi_{m-1}, \mathbf{b}_1) = (\mathbf{T}, \mathbf{b})\}| \tag{5}$$

and the conditional probability given $\mathbf{b}_2$ is

$$\frac{1}{m!}\,|\{\pi_m : \mathcal{A}(\pi_m, \mathbf{b}_2) = (\mathbf{T}, \mathbf{b})\}| \tag{6}$$

From (4), we have (5)=(6), which is the result of this lemma for $\mathbf{b}_1$ and $\mathbf{b}_2$.

3. Given a π_{m-1}, we associate it with m permutations of m elements. For each $1 \le j \le m$, let:

$$\pi_{m,j}(i) = \begin{cases} \text{incr}(\pi_{m-1}(i)) & \text{if } i < j, \\ v & \text{if } i = j, \\ \text{incr}(\pi_{m-1}(i-1)) & \text{otherwise,} \end{cases} \tag{7}$$

$$\text{where} \qquad \text{incr}(x) = \begin{cases} x & \text{if } x < v, \\ x+1 & \text{otherwise.} \end{cases} \tag{8}$$

In other words, we can associate π_{m-1} to m permutations of m elements by inserting v into m possible places in π_{m-1}, and incrementing items larger than v by one. For example, given the permutation $\langle 3, 2, 1, 4\rangle$ which generates a tree with the 2^{nd} and 3^{rd} leaf nodes removed, $v = 3$. Upon inserting v in between 2 and 1 in the permutation, the resultant permutation is $\langle 4, 2, 3, 1, 5\rangle$.

4. We claim that π_{m-1} and $\pi_{m,j}$, for each j, all lead to the same redacted tree. That is, for any j

$$\mathcal{A}(\pi_{m-1}, \mathbf{b}_1) = \mathcal{A}(\pi_{m,j}, \mathbf{b}_2) \tag{9}$$

First, we make an observation regarding the set splitting process in Step 4 of `TreePermGen`. Recall that after the set of redacted leaves and all their ancestors are deleted from the tree, the result is a sequence of subtrees, denoted by $\mathbf{T}$. During the tree generation process, we can incrementally determine the set $\mathbf{T}$ by keeping track of the splitting process. Let us denote by node(A) the node associated with the set A during the execution of `TreePermGen`. When we split a set A into $A_{\texttt{left}}$ and $A_{\texttt{right}}$ in Step 4(b), there are 3 cases to consider:

1. **C1**: Both $A_{\texttt{left}}$ and $A_{\texttt{right}}$ do not contain any items[8] that will be removed. Thus node(A) is not an ancestor of a removed leaf.
2. **C2**: Only one of $A_{\texttt{left}}$ and $A_{\texttt{right}}$ contains items to be removed. Then node(A) is an ancestor of a removed leaf. Suppose $A_{\texttt{left}}$ contains items to be removed, then the subtree with node$(A_{\texttt{right}})$ as root must be in $\mathbf{T}$. We say that A_right contributed to $\mathbf{T}$.
3. **C3**: Both $A_{\texttt{left}}$ and $A_{\texttt{right}}$ contains items to be removed. Then A contains items to be removed, and the subtree rooted at node(A) will never be in $\mathbf{T}$.

By considering the 3 distinct cases above during each step of `TreePermGen`, we can determine the subtrees that appear in $\mathbf{T}_1$ for $\mathcal{A}(\pi_{m-1})$and $\mathbf{T}_2$ for $\mathcal{A}(\pi_{m,j})$. To prove (9), we compare both processes on π_{m-1} and $\pi_{m,j}$ in parallel, and note the subtrees that are added to $\mathbf{T}_1$ and $\mathbf{T}_2$. Let $\mathcal{A}(\pi_{m-1})(i)$ and $\mathcal{A}(\pi_{m,j})(i)$ represent the value of $\pi_{m-1}(i)$ and $\pi_{m,j}(i)$ respectively during the i-th iteration of Step 4 in `TreePermGen`.

5. There are two cases to consider here:

1. v appears after $v+1$ in $\pi_{m,j}$, i.e. $\pi^{-1}_{m,j}(v) > \pi^{-1}_{m,j}(v+1)$ (See Figure 2).
2. v appears before $v+1$ in $\pi_{m,j}$ (See Figure 3).

A *useful insight* in this proof is that for any i, if $\pi_{m-1}(i) < v$, it must be that $\pi_{m,j}(i) < v$ and $\pi_{m,j}(i) = \pi_{m-1}(i)$, and both $\mathcal{A}(\pi_{m-1})$ and $\mathcal{A}(\pi_{m,j})$ will split sets at the same offset from the left-most item. For any i such that $\pi_{m-1}(i) \geq v$, it must be that $\pi_{m,j}(i) = \pi_{m-1}(i) + 1$. This follows from (8), and both $\mathcal{A}(\pi_{m-1})$ and $\mathcal{A}(\pi_{m,j})$ will split sets at the same offset from the right-most item.

Case 1, v appears after $v+1$ in $\pi_{m,j}$:

Let us visualize and compare in parallel $\mathcal{A}(\pi_{m-1})$ and $\mathcal{A}(\pi_{m,j})$, with reference to Figure 2. The two processes are the same (contributing identically to $\mathbf{T}_1$ and $\mathbf{T}_2$), up till before $\mathcal{A}(\pi_{m,j})(i) = v+1$. We can convince ourselves so since if $\mathcal{A}(\pi_{m-1})$ adds a node(A) to $\mathbf{T}_1$ for some set A, due to condition **C2**, the same condition **C2** also exists in $\mathcal{A}(\pi_{m,j})$, and $\mathcal{A}(\pi_{m,j})$ will add node(A) to $\mathbf{T}_2$.

When $\mathcal{A}(\pi_{m,j})(i) = v+1$, we have $\mathcal{A}(\pi_{m-1})(i) = v$, and the two sets created by $\mathcal{A}(\pi_{m-1})$ and $\mathcal{A}(\pi_{m,j})$ both contains items that will be removed. Thus $\mathbf{T}_1$ and $\mathbf{T}_2$ are unaffected. The processes continue till before $\mathcal{A}(\pi_{m,j}) = v$, and both processes will contribute identically to $\mathbf{T}_1$ and $\mathbf{T}_2$. When $\mathcal{A}(\pi_{m,j}) = v$, we pause $\mathcal{A}(\pi_{m-1})$. $\mathcal{A}(\pi_{m,j})$ will split the set with v into two sets, one with v and the other which is the singleton $\{v+1\}$. This step contributes no new subtrees to $\mathbf{T}_2$. From now on, both processes will be identical again. Thus $\mathbf{T}_1$ and $\mathbf{T}_2$ are identical.

[8] More precisely, they do not contain the locations of the objects to be removed, since each value i in the set refers to the i-th leaf node.

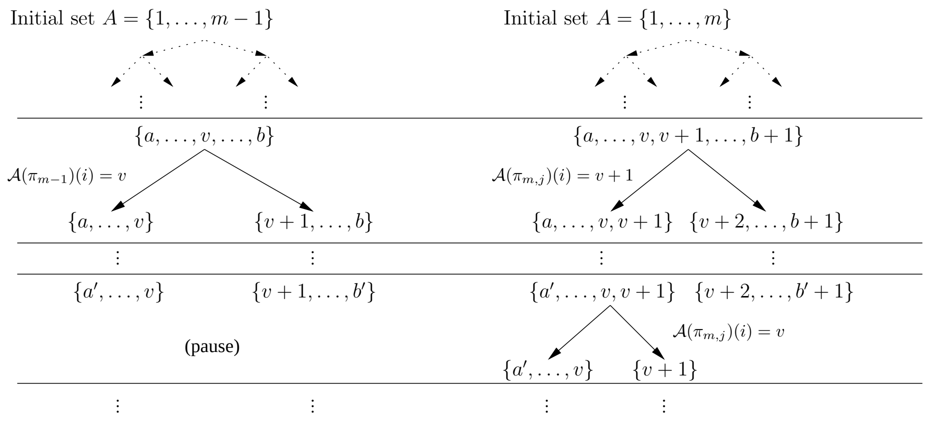

Initial set $A = \{1, \ldots, m-1\}$
Initial set $A = \{1, \ldots, m\}$
$\{a, \ldots, v, \ldots, b\}$
$\mathcal{A}(\pi_{m-1})(i) = v$
$\{a, \ldots, v\}$
$\{v+1, \ldots, b\}$
$\{a, \ldots, v, v+1, \ldots, b+1\}$
$\mathcal{A}(\pi_{m,j})(i) = v+1$
$\{a, \ldots, v, v+1\}$
$\{v+2, \ldots, b+1\}$
$\{a', \ldots, v\}$
$\{v+1, \ldots, b'\}$
(pause)
$\{a', \ldots, v, v+1\}$
$\{v+2, \ldots, b'+1\}$
$\mathcal{A}(\pi_{m,j})(i) = v$
$\{a', \ldots, v\}$
$\{v+1\}$

Fig. 2. Case 1: v appears after $v+1$ in $\pi_{m,j}$.

Case 2, v appears before $v+1$ in $\pi_{m,j}$:

Again, we compare both processes $\mathcal{A}(\pi_{m-1})$ and $\mathcal{A}(\pi_{m,j})$ in parallel, with reference to Figure 3. At every step prior to $\pi_{m,j}(i) = v$, $\mathbf{T}_1$ and $\mathbf{T}_2$ remain identical. When $\mathcal{A}(\pi_{m,j})(i) = v$, we pause $\mathcal{A}(\pi_{m-1})$. $\mathcal{A}(\pi_{m,j})$ creates two new sets, one containing v and one containing $v+1$, as shown in Figure 3. Both sets will not contribute to $\mathbf{T}_2$.

We continue the execution of both processes up till before $\mathcal{A}(\pi_{m-1})(i) = v$ and $\mathcal{A}(\pi_{m,j})(i) = v+1$. In these steps, we can infer that if $\mathcal{A}(\pi_{m-1})$ contribute subtrees to $\mathbf{T}_1$, $\mathcal{A}(\pi_{m,j})$ will also contribute the same subtrees to $\mathbf{T}_2$. Finally, $\mathcal{A}(\pi_{m-1})(i) = v$ and $\mathcal{A}(\pi_{m,j})(i) = v+1$. Now $\mathcal{A}(\pi_{m,j})$ splits the set with $v+1$ into the singleton set $\{v+1\}$ and the set with $v+2$. $\mathcal{A}(\pi_{m-1})$ likewise splits the set with v into two sets. Observe that the right child set of both processes are identical, that $\{v+1\}$ will never contribute to T, and that both processes have identical sets containing v. Thus for the rest of both $\mathcal{A}(\pi_{m-1})$ and $\mathcal{A}(\pi_{m,j})$, $\mathbf{T}_1$ and $\mathbf{T}_2$ remain identical.

6. There are m ways to insert w into π_{m-1}. Thus, for any π_{m-1}, there are m permutation π_m that leads to the same redacted tree. For a fixed $1 \leq v \leq m$, (5) and (6) together maps a (π_{m-1}, v, j) for any $1 \leq j \leq m$, to exactly one π_m, and every π_m is mapped to. Thus (5) and (6) together is a bijective mapping and this process will generate all possible π_m from all possible π_{m-1}, and we have the equality (4) and the result for $\mathbf{b}_1$ and $\mathbf{b}_2$. We can generalise the claim to any $\mathbf{b}_1$ and $\mathbf{b}_2$ that differs in more than one removed substrings and by more than one item each, by repeatedly applying the argument here on just one removed substring and holding the other removed substrings constant. $\square$

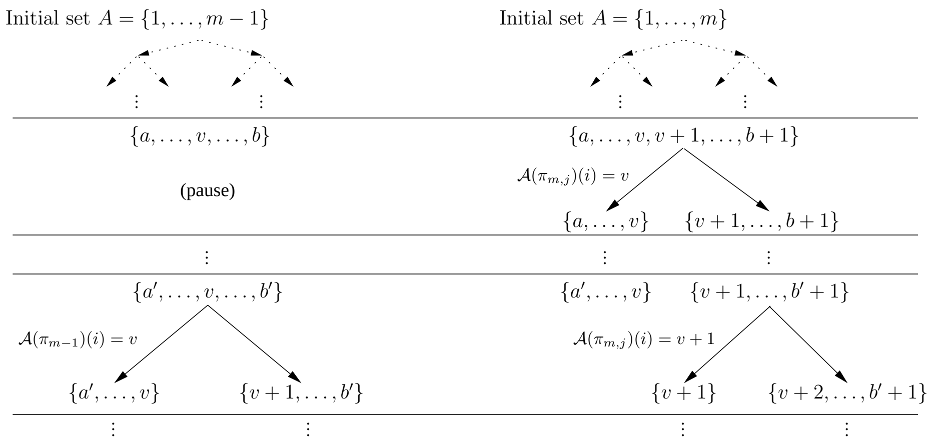

Initial set $A = \{1, \ldots, m-1\}$
Initial set $A = \{1, \ldots, m\}$
$\{a, \ldots, v, \ldots, b\}$
(pause)
$\{a, \ldots, v, v+1, \ldots, b+1\}$
$\mathcal{A}(\pi_{m,j})(i) = v$
$\{a, \ldots, v\}$
$\{v+1, \ldots, b+1\}$
$\{a', \ldots, v, \ldots, b'\}$
$\mathcal{A}(\pi_{m-1})(i) = v$
$\{a', \ldots, v\}$
$\{v+1, \ldots, b'\}$
$\{a', \ldots, v\}$
$\{v+1, \ldots, b'+1\}$
$\mathcal{A}(\pi_{m,j})(i) = v+1$
$\{v+1\}$
$\{v+2, \ldots, b'+1\}$

Fig. 3. Case 2: v appears before $v+1$ in π_m.

E Security of S5

Unforgeability. From the definition of cryptographic secure pseudo random number generator and Theorem 2, we conclude that S5 is secure.

Corollary 5 *For any positive polynomials (in κ) t and q,* S5 *is $(t, q, \frac{\epsilon_1}{1-\epsilon_2})$-secure against existential forgeries with respect to $\succ$, if* SSS *is $(t + qt_0, q, \epsilon_1)$-secure against existential forgeries with respect to $\supseteq$, $\mathcal{H}$ is $(t + qt_{\mathtt{S5}}, \epsilon_2)$-collision-resistant, and G is a cryptographic secure pseudo random number generator, where t_0 is the running time of $\mathcal{H}$, $t_{\mathtt{S5}}$ is the time needed by* S5 *to sign a document, and κ is the security parameter.*

Privacy. From the definition of cryptographic secure pseudo random number generator and and Theorem 3, we conclude that S5 is privacy preserving.

Corollary 6 *The redactable signature scheme* S5 *is privacy preserving (as defined in Definition 3), assuming that the hash function h satisfies the property: for any integer constants $a, b \geq 1$, the two distributions $\mathcal{X} = g^{h(U_1)h(U_2)\ldots h(U_a)} \bmod n$ and $\mathcal{Y} = g^{h(U'_1)h(U'_2)\ldots h(U'_b)} \bmod n$ are computational indistinguishable, and G is a cryptographic secure pseudo random number generator, where n is a RSA modulus, g is an element of large order in $\mathbb{Z}_n^*$ and U_i's and U'_j's are all independent uniform random variables over $\mathbb{Z}_n$, and $h(\cdot)$ is used to define $\mathcal{H}$ in Table 1.*

We found if $h(U)$ is indistinguishable from the uniform random input U, then h satisfies the property in Corollary 6.