6-2020

Editing-enabled signatures: A new tool for editing authenticated data

Binanda SENGUPTA
Singapore Management University, binandas@smu.edu.sg

Yingjiu LI
University of Oregon

Yangguang TIAN
Singapore Management University, ygtian@smu.edu.sg

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Published in IEEE Internet of Things, 2020 June, 7 (6), 4997-5007
https://doi.org/10.1109/10.1109/JIOT.2020.2972741

Editing-Enabled Signatures: A New Tool for Editing Authenticated Data

Binanda Sengupta, Yingjiu Li, *Member, IEEE*, Yangguang Tian, and Robert H. Deng, *Fellow, IEEE*

***Abstract*—Data authentication primarily serves as a tool to achieve data integrity and source authentication. However, traditional data authentication does not fit well where an intermediate entity (editor) is required to modify the authenticated data provided by the source/data owner before sending the data to other recipients. To ask the data owner for authenticating each modified data can lead to higher communication overhead. In this work, we introduce the notion of *editing-enabled signatures* where the data owner can choose *any* set of modification operations applicable on the data and still can restrict any possibly untrusted editor to authenticate the data modified using an operation from this set *only*. Moreover, the editor does not need to interact with the data owner in order to authenticate the data every time it is modified. We construct an editing-enabled signature scheme that derives its efficiency from mostly lightweight cryptographic primitives. We formalize the security model for editing-enabled signatures and analyze the security of our editing-enabled signature scheme. Editing-enabled signatures can find numerous applications that involve generic editing tasks and privacy-preserving operations. We demonstrate how our editing-enabled signature scheme can be applied in two privacy-preserving applications.**

***Index Terms*—Editing-enabled signatures, editing-functions, hash-chains, privacy-preserving applications.**

I. Introduction

UBIQUITOUS computing [1] and the Internet of Things (IoT) [2] enable embedded devices to perform certain computations and to communicate the data produced by these computations to back-end services hosted by remote clouds. For example, wearable fitness trackers (or patient monitoring devices) collect data from the persons wearing them and send the data to a cloud server that provides utility services. For IoT applications, resource constraints and lack of security in these devices are two major concerns [3], [4]. These low-power devices are often wireless and restricted to be connected only with a nearby smartphone that in turn communicates with the cloud server via the Internet. This is a typical example of the eMbedded-Gateway-Cloud (MGC) model where the smartphone acts as a field gateway connecting the embedded devices to the cloud sever [5], [6]. In addition, the smartphone can perform certain computations on the collected data before sending the modified data to the cloud server. However, a smartphone is prone to attacks due to malwares that can corrupt the computations and provide erroneous data to the server. It is thus imperative to detect if the smartphone has correctly performed the computations on the original data.

B. Sengupta, Y. Tian and R. H. Deng are with the School of Information Systems, Singapore Management University, Singapore. Y. Li is with the Computer and Information Science Department, University of Oregon, USA. *Corresponding author: Binanda Sengupta.*

E-mail: {binandas,ygtian,robertdeng}@smu.edu.sg, yingjiul@uoregon.edu.

A similar situation arises when a surveillance camera (e.g., an IoT camera in a car/aerial drone, or a CCTV camera) captures an image/video, and a publisher (e.g., a media house) publishes the image/video file as an evidence for an incident (e.g., a traffic accident/criminal activity). Such a file often contains sensitive parts (e.g., human faces) that need to be protected due to privacy concerns. This requires the publisher to modify the file in order to hide these parts (e.g., by blurring them) before publishing it [7], [8]. However, an untrusted publisher can modify the file maliciously for various reasons. Thus, a receiver of the modified file should be able to check if the publisher has modified the original file correctly.

For many IoT applications, the traffic is unidirectional in that resource-constrained end-devices can only send data to gateways to modify the data. So, it is not possible to send the modified data back to the end-devices and ask them to authenticate the data for every modification. Even for applications that involve enough resources it is inefficient, due to communication overhead, to request the data owner for authentication on each modified data.

Problem statement. We generalize the aforementioned scenarios to the following problem: a *data owner* delegates her data to any intermediate entity so that this entity can perform certain modification operations on the data and provide the modified data to a third-party *receiver*. We call this intermediate entity an *editor*, who can edit the data, and a modification operation applicable on the data an *editing-function*. The data owner specifies a set of editing-functions, and the editor is allowed to apply an editing-function from that set only. However, in case the editor is untrusted, she can choose an illegitimate editing-function, or incorrect data, or both. Thus, the receiver should be able to verify the *authenticity* of the modified data (i.e., if it has been derived by applying a legitimate editing-function on the original data provided by the data owner). Moreover, the data owner should be able to select *any* editor after she authenticates the data. It is also desired that the editor does not need to ask the data owner to authenticate the derived data for every modification.

Why are existing techniques not suitable? Traditional digital signatures generated by the data owner on the original data do not meet the authenticity requirements stated above, because a possible modification in the underlying data invalidates the data owner's signatures on the original data. One possible way is that the editor asks the data owner for a signature on the derived data after every modification she performs on the data. However, this solution requires one round of communication between the data owner and the editor for *each* such modification — which makes the solution

inefficient. Moreover, as we have mentioned above, many IoT applications do not support such interactions.

There exist other specialized signatures such as sanitizable signatures [9], [10] and redactable signatures [11], [12] that enable an intermediate entity to modify the data given by its owner. Sanitizable signatures permit an entity (sanitizer) to *arbitrarily* modify the owner's data where, to the best of our knowledge, the data owner *cannot* restrict the set of allowed editing-functions. Moreover, the signing algorithm takes the public key of the sanitizer as input, so that only the sanitizer having the secret key can modify the data. However, in our problem statement, the data owner has the flexibility to choose any editor even after she authenticates the data. On the other hand, redactable signatures handle deletion operations *only* — this makes the set of allowed editing-functions too limited.

Notion of editing-enabled signatures. In this work, we introduce *editing-enabled signatures* which serve as a solution to the problem mentioned above. A data owner in an editing-enabled signature (EES) scheme generates a signature on the data in such a way that *any* editor can later modify the data based on a set of *permissible* editing-functions in an *authenticated* fashion. Editing-enabled signatures are very flexible and very restrictive at the same time: the data owner can specify any set of editing-functions of her choice and she can still restrict the editor to modify the data using editing-functions from that set only. In addition, the editor does *not* need to interact with the data owner to authenticate the edited data — which meets the IoT requirements mentioned above.

Our construction: Techniques and merits. Following the notion of editing-enabled signatures, we construct an EES scheme using the following techniques.

- An important building block of our EES scheme is the way we represent the set of editing-functions applicable on the data. As an editing-function can be complex in nature, we consider an editing-function as a *sequence* of basic functions (or simply, functions) (see Section II-A). The data owner authenticates the output of such a sequence of functions using traditional *signatures*.
- Representing an editing-function as a sequence of functions is helpful when every subsequence of the sequence is an editing-function (see Section IV-A). However, authenticating the output of such a sequence does not guarantee the authenticity of the outputs of all subsequences. Signing each of these outputs demands many public-key operations to be performed by the (possibly resource-constrained) data owner. Instead, in our construction, we use a *collision-resistant hash-chain* to bind the outputs in an authenticated way. The data owner signs the hash corresponding to the output of the sequence — which ensures the authenticity of the outputs of all subsequences.
- In our construction, the representation of editing-functions is realized using a *function tree*, where each edge represents a function and each path starting from the root-node and ending at some node of the tree represents an editing-function. The function tree is made public so that a receiver knows the exact set of editing-functions. Given the modified data and the corresponding editing-function, the receiver computes the *final hash value* of the respective hash-chain and verifies it using the signature generated by the data owner and provided by the editor. In case the function tree has two or more leaf-nodes, the data owner needs to authenticate the output of each sequence using a hash-chain and a signature. We reduce the number of signatures required by building a *Merkle hash tree* [13] over the final hash values of all hash-chains and signing its root-digest *only*.

Our EES construction provides the data owner with the flexibility to specify the set of editing-functions of her choice. The editor, on the other hand, is restricted to edit the data by applying editing-functions from that set only. Our construction involves mostly lightweight cryptographic hash functions — which makes it suitable for resource-constrained IoT devices.

Significance of our contribution. Editing-enabled signatures can find many applications where a data owner allows an editor to modify the data before publishing the data to an intended receiver and the receiver can verify if the possibly untrusted editor has modified the data as specified by the data owner. For example, the data owner might want the editor to perform certain *generic editing tasks* on a data file (e.g., to embed copyright-related information in an image) before uploading the modified file to some storage server. Editing-enabled signatures can also be used in *privacy-preserving applications* where the data owner does not want to disclose certain sensitive parts of the data. The receiver, without an access to these sensitive parts, can still check if the editor has edited the original data correctly. Among numerous potential applications, we consider privacy-preserving image/video publishing and privacy-preserving data publishing in this paper.

Organization of the paper. The rest of the paper is organized as follows. Section II introduces the notion and security model of editing-enabled signatures. In Section III, we construct an editing-enabled signature (EES) scheme and analyze its security. In Section IV, we discuss the efficiency and an extension of our EES scheme. Section V describes applications of editing-enabled signatures. We summarize the related work in Section VI and conclude the paper in Section VII.

Notation used in the paper. For two integers a and b (where $a \leq b$), the set $\{a, a+1, \ldots, b\}$ is denoted by $[a, b]$ as well. λ is the security parameter. An algorithm $\mathcal{A}(1^\lambda)$ taking λ as input is called a *probabilistic polynomial-time* (PPT) algorithm if: 1) its running time is polynomial in λ, and 2) its output is a random variable that depends on the internal coin tosses of $\mathcal{A}$. We use $\mathcal{A}^{\mathcal{O}}$ to denote that $\mathcal{A}$ is allowed to query an oracle $\mathcal{O}$. A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* in λ if, for all positive integers c and for all sufficiently large λ, we have $f(\lambda) < \frac{1}{\lambda^c}$.

II. Overview of Editing-Enabled Signatures

In this section, we introduce the notion of editing-enabled signatures for a data block. Editing-enabled signatures allow a data owner to split her data file into multiple data blocks and specify (possibly) different operations for different blocks. An editing-enabled signature (EES) scheme involves three entities: a data owner $\mathcal{D}$, an editor $\mathcal{E}$ and a receiver $\mathcal{R}$. $\mathcal{D}$ signs a data block for a set of editing-functions and delegates the task of editing to $\mathcal{E}$. Later, $\mathcal{E}$ derives a data block by applying an

editing-function on the data block. Given a derived data block and the signature, $\mathcal{R}$ can verify if $\mathcal{E}$ has correctly edited the data block provided by $\mathcal{D}$.

A. Editing-Functions

The data owner chooses a set of editing-functions permissible for a data block. Each editing-function can be a composition of multiple basic functions. In order to distinguish them, we denote the set of *basic functions* (or simply, *functions*) by $\mathtt{F}$ and the set of *editing-functions* by $\overline{\mathtt{F}}$. We assume that each function is *deterministic* and *unary*, and it can be computed in *polynomial* time. Based on the hardness of inverting the outputs, a function in $\mathtt{F}$ can be classified as *one-way* (hard to invert) or *two-way* (easy to invert). One-way functions can be useful for privacy-preserving applications.

An editing-function can be composed of *one or more* (*polynomially many*) functions. Thus, an editing-function is also deterministic, unary and computable in polynomial time. We further assume that $|\overline{\mathtt{F}}|$ is polynomial and thus the output of $\overline{\mathtt{F}}$, when applied on a data block, can also be computed in polynomial time. We say that *an editing-function (or equivalently, a sequence of functions) is applied on a data block* in order to indicate the following: the first function of the sequence is applied on the data block to get an *intermediate* output which is provided as input to the second function of the sequence, and so on. The output of the last function of the sequence is the (*final*) output — we call this output the *derived data block* corresponding to the editing-function. As the editor may also provide the original (i.e., unedited) data block to the receiver, we assume that $\mathtt{F}$ includes the *identity function* (say, f_I), and $\overline{\mathtt{F}}$ includes the sequence $\{f_I\}$.

B. Editing-Enabled Signatures

As we have mentioned earlier, we consider editing-enabled signatures for a data block. A data block is identified by a unique block-identifier $\mathtt{id}$. We use the term "data block m" to denote a data block where a message m is stored in the block. For example, we can consider a relational database table to be a data block identified by a unique table-identifier (that serves as a block-identifier) and the content of the table to be the message. However, the message present in a data block can vary (e.g., when the table is updated). So, the block-identifier alone does not suffice to uniquely identify a data block m. We use a message-identifier $\mathtt{mid}$ which, along with the block-identifier, uniquely identifies the message stored in that block. For example, given a table identified by $\mathtt{id}$, a timestamp (or a counter denoting the number of updates) can serve as $\mathtt{mid}$, such that the content of the table at time $\mathtt{mid}$ (or after $\mathtt{mid}$ number of updates) is fixed. Throughout the rest of the paper, we mention a data block m to be identified by ($\mathtt{id}$, $\mathtt{mid}$) in order to denote that the message m, corresponding to a message-identifier $\mathtt{mid}$, is stored in the data block identified by $\mathtt{id}$. We define an EES scheme as follows. The workflow of an EES scheme is shown in Figure 1.

Definition 1 (Editing-Enabled Signatures): An EES scheme consists of the following algorithms.

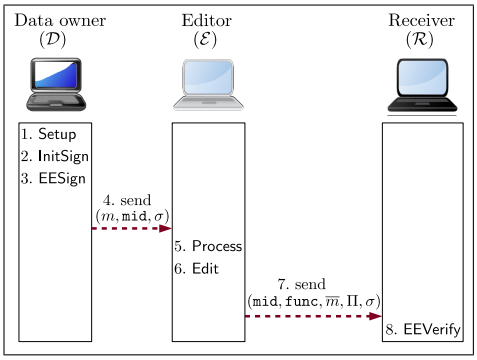


Fig. 1: Workflow of an EES scheme.

- $\mathsf{Setup}(1^\lambda, \mathtt{id}, \mathtt{F})$: The algorithm generates a secret key-public key pair (sk, pk). Given a data block identified by $\mathtt{id}$ and a set of functions $\mathtt{F}$, it decides a set of permissible editing-functions $\overline{\mathtt{F}}$ and specifies the set $\overline{\mathtt{F}}$ by a parameter $\mathtt{spec}$. Finally, the algorithm outputs sk and a public parameter $\mathtt{pp}$ that includes pk, $\mathtt{id}$, $\mathtt{spec}$.
- $\mathsf{InitSign}(m, \mathtt{pp})$: Given a data block identified by $\mathtt{id}$ and a message m, the algorithm selects a message-identifier $\mathtt{mid}$ such that the data block m is uniquely identified by the pair $(\mathtt{id}, \mathtt{mid})$. The algorithm outputs $\mathtt{mid}$.
- $\mathsf{EESign}(m, \mathtt{mid}, sk, \mathtt{pp})$: Given a data block m identified by $(\mathtt{id}, \mathtt{mid})$, a secret key sk and a set of editing-functions specified by $\mathtt{spec}$, the algorithm produces an editing-enabled signature σ. The algorithm outputs σ.
- $\mathsf{Process}(m, \mathtt{mid}, \sigma, \mathtt{pp})$: Given a data block m identified by $(\mathtt{id}, \mathtt{mid})$, a signature σ and a public parameter $\mathtt{pp}$, the algorithm processes the data block m based on $\mathtt{spec}$. The algorithm outputs $\perp$ or a helper metadata $\mathtt{help}$ that later helps to edit the data block according to an editing-function.
- $\mathsf{Edit}(m, \mathtt{mid}, \sigma, \mathtt{help}, \mathtt{func}, \mathtt{pp})$: Given an editing-function specified by a metadata $\mathtt{func}$ and a helper metadata $\mathtt{help}$, the algorithm generates a derived data block $\overline{m}$. It outputs the derived data block $\overline{m}$ and a verification metadata Π.
- $\mathsf{EEVerify}(\overline{m}, \mathtt{mid}, \sigma, \Pi, \mathtt{func}, \mathtt{pp})$: The algorithm verifies if the editing-function specified by $\mathtt{func}$ belongs to $\overline{\mathtt{F}}$ (specified by $\mathtt{spec}$) and if σ is a valid signature corresponding to the derived data block $\overline{m}$. The algorithm outputs 0 if any of the verification fails; the algorithm outputs 1, otherwise.

C. Security Model

The editor $\mathcal{E}$ in an EES scheme may be untrusted in that: given a set of editing-functions specified by $\mathtt{spec}$ and a data block m provided by $\mathcal{D}$, $\mathcal{E}$ may attempt to authenticate (on behalf of $\mathcal{D}$) the output of an illegitimate editing-function (that is not specified in $\mathtt{spec}$), or a message m' not provided by $\mathcal{D}$, or both. In EES, the (possibly untrusted) editor is $\mathcal{E}$ modeled as the probabilistic polynomial-time (PPT) adversary $\mathcal{A}$.

We say that two editing-functions (or two sequences of functions) are *equivalent* with respect to a data block m, if the

sequence of outputs for one of them (including the intermediate and final outputs) is exactly same as that for another. In the following security model, we exclude equivalent functions for a given data block since they produce the same editing results. We describe the security game between a challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ as follows.

- **Setup phase.** $\mathcal{C}$ generates a secret key-public key pair (sk, pk). Given a data block, $\mathcal{C}$ selects a unique identifier $\mathtt{id}$ and a set of deterministic functions $\mathtt{F}$. $\mathcal{C}$ runs Setup with these inputs to generate a secret key-public key pair (sk, pk) and public parameter $\mathtt{pp}$. $\mathcal{C}$ sends $\mathtt{pp}$ to $\mathcal{A}$.
- **Query phase.** $\mathcal{C}$ responds to the signing queries made by $\mathcal{A}$ as follows. $\mathcal{A}$ adaptively chooses a sequence of pairs $Q = \{(\mathtt{mid}_j, m_j)\}_{1 \leq j \leq q_1}$ for the data block m_j identified by the pair $(\mathtt{id}, \mathtt{mid}_j)$ and sends Q to $\mathcal{C}$, where q_1 is a polynomial in the security parameter λ. For each j, $\mathcal{C}$ runs EESign to generate a signature σ_j on the data block m_j for $\mathtt{spec}$ and gives σ_j to $\mathcal{A}$.
- **Challenge phase.** $\mathcal{A}$ attempts to generate a valid signature for a message that is the output of an editing-function, not necessarily from $\mathtt{spec}$, when applied on the data block for a message-identifier which is not used in the query phase. $\mathcal{A}$ may also aim to generate a valid signature for a message that is not the output of an editing-function from $\mathtt{spec}$ when applied on the data block for a message-identifier used in the query phase. In particular, $\mathcal{A}$ may take any of the following actions in order to win the security game.
  - $\mathcal{A}$ selects a data block m identified by the pair $(\mathtt{id}, \mathtt{mid})$ such that $\mathtt{mid} \neq \mathtt{mid}_j$ for any $1 \leq j \leq q_1$. $\mathcal{A}$ also selects an editing-function specified by the metadata $\mathtt{func}$ which may *not* be present in $\mathtt{spec}$ designated by $\mathcal{C}$. $\mathcal{A}$ sends $\mathtt{func}$, $\mathtt{mid}$, a derived data block $\overline{m}$, a verification metadata Π and a signature σ to $\mathcal{C}$. $\mathcal{C}$ runs $\mathsf{EEVerify}$ on these inputs. $\mathcal{A}$ wins the security game if $\mathsf{EEVerify}$ outputs 1.
  - $\mathcal{A}$ selects a data block m_j identified by the pair $(\mathtt{id}, \mathtt{mid}_j)$ for some $1 \leq j \leq q_1$. $\mathcal{A}$ also selects two editing-functions specified by the metadata $\mathtt{func}$ and $\mathtt{func}'$, respectively, such that the first one is specified in $\mathtt{spec}$, the second one is not specified in $\mathtt{spec}$ and they are not equivalent. $\mathcal{A}$ sends $\mathtt{func}$, $\mathtt{mid}_j$, the derived data block $\overline{m}'$ (corresponding to $\mathtt{func}'$), a verification metadata Π and a signature σ to $\mathcal{C}$. $\mathcal{C}$ runs $\mathsf{EEVerify}$ on these inputs. $\mathcal{A}$ wins the security game if $\mathsf{EEVerify}$ outputs 1.

Definition 2 (Security of an EES Scheme): An EES scheme is secure if, for any PPT adversary $\mathcal{A}$, the adversary $\mathcal{A}$ wins the security game described above only with a probability negligible in the security parameter λ.

III. An Editing-Enabled Signature Scheme

Our editing-enabled signature (EES) scheme consists of the following algorithms. For the ease of presentation, we include the respective entity, that executes an algorithm, in the description of that algorithm.

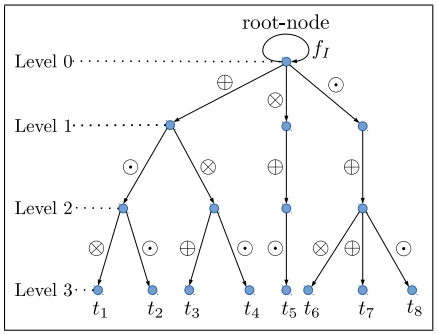


Fig. 2: An example of a function tree $\mathcal{T}_{\mathtt{id},f}$ with $n = 8$ leaf-nodes $\{t_1, t_2, \ldots, t_8\}$ and $L = 4$ levels $\{0, 1, 2, 3\}$.

- $\mathsf{Setup}(1^\lambda, \mathtt{id}, \mathtt{F})$: The data owner $\mathcal{D}$ decides an EUF-CMA (existentially unforgeable under adaptive chosen-message attacks [14]) digital signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ and a collision-resistant hash function H. $\mathcal{D}$ runs $\Sigma.\mathsf{KeyGen}(1^\lambda)$ to generate a secret key-public key pair (sk, pk). Let $\mathcal{I}_B$ and $\mathcal{I}_M$ be the space of block-identifiers and the space of message-identifiers, respectively.
  Given a set of functions $\mathtt{F}$ and a data block identified by a unique identifier $\mathtt{id} \in \mathcal{I}_B$, $\mathcal{D}$ decides the set of editing-functions $\overline{\mathtt{F}}$ for the data block and generates a function tree $\mathcal{T}_{\mathtt{id},f}$. We note that $\mathcal{T}_{\mathtt{id},f}$ specifies the editing-functions the editor $\mathcal{E}$ can apply on the data block, and each editing-function is a sequence of functions chosen from $\mathtt{F}$. $\mathcal{T}_{\mathtt{id},f}$ consists of a *root-node* and n *leaf-nodes* ($n \geq 1$). All other nodes present in $\mathcal{T}_{\mathtt{id},f}$ are called *intermediate nodes*. The path $\mathtt{path}_i$ from the root-node to the i-th leaf-node t_i is denoted by a sequence of functions $\mathtt{seq}_i$, where each function in $\mathtt{seq}_i$ denotes an edge in $\mathcal{T}_{\mathtt{id},f}$ and is chosen from $\mathtt{F}$. For simplicity, we assume that $\mathcal{T}_{\mathtt{id},f}$ consists of L levels, where the root-node is at level $l = 0$, all leaf-nodes are at level $l = L - 1$ and an intermediate node resides at a level $l \in [1, L-2]$. In other words, $|\mathtt{seq}_1| = |\mathtt{seq}_2| = \cdots = |\mathtt{seq}_n| = L - 1$. Figure 2 shows a function tree for $n = 8$, $L = 4$ and $\mathtt{F} = \{\oplus, \otimes, \odot, f_I\}$, where f_I is the identity function and $\oplus, \otimes, \odot$ are some functions applicable on the data block. For each leaf-node t_i ($1 \leq i \leq n$) of $\mathcal{T}_{\mathtt{id},f}$, we denote the functions in the sequence $\mathtt{seq}_i$ (from the root-node to t_i) by $f_{i,1}, f_{i,2}, \ldots, f_{i,L-1}$ each of which is chosen from $\mathtt{F}$ (e.g., $f_{4,1} = \oplus, f_{4,2} = \otimes, f_{4,3} = \odot$ for $\mathtt{seq}_4$). *Each subsequence* (starting from the root-node) of $\mathtt{seq}_i = \{f_{i,1}, f_{i,2}, \ldots, f_{i,L-1}\}$ represents an *editing-function*, i.e., each of the sequences $\{f_{i,1}\}, \{f_{i,1}, f_{i,2}\}, \ldots, \{f_{i,1}, f_{i,2}, \ldots, f_{i,L-1}\}$ is an editing-function. Each editing-function can be specified by *at least one pair* (i, l) for some $i \in [1, n]$ and a unique level l of $\mathcal{T}_{\mathtt{id},f}$ (e.g., $\{\odot, \oplus\}$ can be specified by $(6, 2)$ or $(7, 2)$ or $(8, 2)$). A function may occur multiple times in $\mathtt{seq}_i$ (e.g., $f_{2,2} = f_{2,3} = \odot$ for $\mathtt{seq}_2$), and a function in $\mathtt{seq}_i$ may be equal to a function in $\mathtt{seq}_j$ for $i \neq j$ (e.g., $f_{1,1} = f_{5,2} = f_{7,3} = \oplus$). Moreover, $\mathtt{path}_i$ and $\mathtt{path}_j$ ($i \neq j$) may share a sub-path starting from

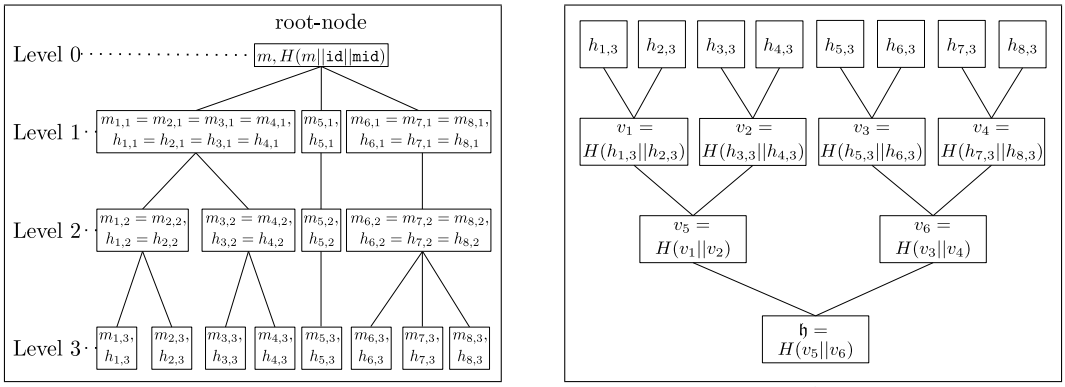


Fig. 3: (a) An editing tree $\mathcal{T}_{\texttt{id,mid},e}$ corresponding to $\mathcal{T}_{\texttt{id},f}$ shown in Figure 2. (b) A Merkle hash tree $\mathcal{T}_{\texttt{id,mid},h}$ built over the hash values stored at the leaf-nodes of $\mathcal{T}_{\texttt{id,mid},e}$. The *Merkle proof* for a leaf-node v of $\mathcal{T}_{\texttt{id,mid},h}$ comprises the contents of the siblings of the nodes present on the path from v to the root-node (e.g., the Merkle proof for $h_{7,3}$ consists of $(h_{8,3}, v_3, v_5)$).

the root-node and ending at a level $l \leq L-2$, which implies that $f_{i,1} = f_{j,1}, f_{i,2} = f_{j,2}, \ldots, f_{i,l} = f_{j,l}$ (e.g., $f_{6,1} = f_{8,1} = \odot, f_{6,2} = f_{8,2} = \oplus$ for $\texttt{seq}_6$ and $\texttt{seq}_8$). $\mathcal{D}$ includes the identity function $f_{i,0} = f_I$ for all $i \in [1, n]$ — which denotes that the derived block is the original data block itself.

The data owner $\mathcal{D}$ labels the edges of $\mathcal{T}_{\texttt{id},f}$ by the corresponding functions and includes the labeled function tree $\mathcal{T}_{\texttt{id},f}$ in $\texttt{spec}$. Then, $\mathcal{D}$ includes $pk, \texttt{id}, \texttt{spec}$, the description of H and Σ in the public parameter $\texttt{pp}$.

- $\mathsf{InitSign}(m, \texttt{pp})$: Given a data block identified by $\texttt{id}$ and a message m, the data owner $\mathcal{D}$ selects a message-identifier $\texttt{mid} \in \mathcal{I}_M$ such that the data block m is uniquely identified by the pair $(\texttt{id}, \texttt{mid})$.
- $\mathsf{EESign}(m, \texttt{mid}, sk, \texttt{pp})$: Given a data block m, a message-identifier $\texttt{mid}$, the secret key sk and the parameter $\texttt{spec}$ included in $\texttt{pp}$, the data owner $\mathcal{D}$ generates an editing tree $\mathcal{T}_{\texttt{id,mid},e}$ corresponding to the function tree $\mathcal{T}_{\texttt{id},f}$ included in $\texttt{spec}$ as follows. Let $m_{i,l}$ (for $1 \leq i \leq n$ and $0 \leq l \leq L-1$) denote the data block derived from the original data block m by applying the sequence of functions $\{f_{i,0}, f_{i,1}, f_{i,2}, \ldots, f_{i,l}\}$. That is, $m_{i,l} = f_{i,l}(\cdots(f_{i,1}(f_{i,0}(m)))\cdots)$. We note that $m_{i,0} = m$ for all $i \in [1, n]$. The structure of $\mathcal{T}_{\texttt{id,mid},e}$ is similar to that of $\mathcal{T}_{\texttt{id},f}$; the nodes in $\mathcal{T}_{\texttt{id,mid},e}$ contain the corresponding *derived data blocks and their hash values* (see Figure 3(a)). The root-node of $\mathcal{T}_{\texttt{id,mid},e}$ contains the pair $(m, H(m||\texttt{id}||\texttt{mid}))$, where H is the collision-resistant hash function described in $\texttt{spec}$. For each $1 \leq i \leq n$, the node at level l corresponding to $\texttt{seq}_i$ contains the pair $(m_{i,l}, h_{i,l})$, where

$$h_{i,l} = \begin{cases} H(m||\texttt{id}||\texttt{mid}), & \text{if } l = 0 \\ H(h_{i,l-1}||m_{i,l}||\texttt{id}||\texttt{mid}), & \text{if } 1 \leq l \leq L-2 \\ H(i||h_{i,l-1}||m_{i,l}||\texttt{id}||\texttt{mid}), & \text{if } l = L-1. \end{cases}$$

We denote the path from the root-node to the i-th leaf-node of $\mathcal{T}_{\texttt{id,mid},e}$ by $\texttt{epath}_i$ which corresponds to $\texttt{path}_i$ of $\mathcal{T}_{\texttt{id},f}$. For $\texttt{path}_i$ and $\texttt{path}_j$ ($i \neq j$) sharing a sub-path in $\mathcal{T}_{\texttt{id},f}$ (starting from the root-node and ending at level $l \leq L-2$), the nodes on the corresponding sub-path shared by $\texttt{epath}_i$ and $\texttt{epath}_j$ contain same hash values (e.g., $h_{6,0} = h_{7,0}$, $h_{6,1} = h_{7,1}$, $h_{6,2} = h_{7,2}$).

If $\mathcal{T}_{\texttt{id,mid},e}$ has $n > 1$ leaf-nodes, $\mathcal{D}$ constructs a Merkle hash tree [13] over the *hash values* stored in the leaf-nodes of $\mathcal{T}_{\texttt{id,mid},e}$. Figure 3(b) shows a Merkle hash tree $\mathcal{T}_{\texttt{id,mid},h}$ corresponding to $\mathcal{T}_{\texttt{id,mid},e}$ shown in Figure 3(a). Let $\mathfrak{h}$ be the root-digest of $\mathcal{T}_{\texttt{id,mid},h}$. If $\mathcal{T}_{\texttt{id,mid},e}$ contains only $n = 1$ leaf-node, then $\mathfrak{h} = h_{1,L-1}$. Finally, $\mathcal{D}$ generates a signature $\sigma = \Sigma.\mathsf{Sign}(sk, \mathfrak{h})$ on $\mathfrak{h}$.

- $\mathsf{Process}(m, \texttt{mid}, \sigma, \texttt{pp})$: Given the function tree $\mathcal{T}_{\texttt{id},f}$ and the data block m identified by $(\texttt{id}, \texttt{mid})$, $\mathcal{E}$ constructs $\mathcal{T}_{\texttt{id,mid},e}$ and $\mathcal{T}_{\texttt{id,mid},h}$ following the same way as described above. Finally, $\mathcal{E}$ verifies σ by checking if

$$\Sigma.\mathsf{Verify}(pk, \mathfrak{h}, \sigma) \stackrel{?}{=} 1. \tag{1}$$

If the equality holds, $\mathcal{E}$ stores the signature σ sent by the data owner $\mathcal{D}$ and the helper metadata $\texttt{help} = (\mathcal{T}_{\texttt{id,mid},e}, \mathcal{T}_{\texttt{id,mid},h})$ that later helps her edit the data block according to an editing-function present in $\mathcal{T}_{\texttt{id},f}$. $\mathcal{E}$ outputs $\perp$ in case the equality does not hold.

- $\mathsf{Edit}(m, \texttt{mid}, \sigma, \texttt{help}, \texttt{func}, \texttt{pp})$: Given the metadata $\texttt{help}$ and an editing-function specified by $\texttt{func} = (i, l)$, $\mathcal{E}$ retrieves the derived data block $\overline{m} = m_{i,l} = f_{i,l}(\cdots(f_{i,1}(f_{i,0}(m)))\cdots)$ and the Merkle proof (only for $n > 1$) for $h_{i,L-1}$, by looking up $\mathcal{T}_{\texttt{id,mid},e}$ and $\mathcal{T}_{\texttt{id,mid},h}$. Then, $\mathcal{E}$ includes $h_{i,l-1}$ (or $h_{i,l}$ for $l = 0$) and the Merkle proof in the verification metadata Π.
- $\mathsf{EEVerify}(\overline{m}, \texttt{mid}, \sigma, \Pi, \texttt{func}, \texttt{pp})$: Given the public parameter $\texttt{pp}$ and a metadata $\texttt{func} = (i, l)$, the receiver $\mathcal{R}$ checks if the editing-function corresponding to $\texttt{func}$ is present in $\texttt{spec}$; $\mathcal{R}$ outputs 0, otherwise. Given a derived data block $\overline{m}$, a verification metadata Π, $\texttt{func}$ and $\texttt{pp}$, $\mathcal{R}$ gets the value $h_{i,l}$ from Π (for $l = 0$) or computes $h_{i,l} = H(h_{i,l-1}||\overline{m}||\texttt{id}||\texttt{mid})$ for $l > 0$. For $l = 0$, $\mathcal{R}$ outputs 0 if $h_{i,l} \neq H(\overline{m}||\texttt{id}||\texttt{mid})$. Otherwise, $\mathcal{R}$ computes the sequence $\{(m_{i,l+1}, h_{i,l+1}), (m_{i,l+2}, h_{i,l+2}), \ldots, (m_{i,L-1}, h_{i,L-1})\}$, using $m_{i,l+1} = f_{i,l+1}(\overline{m})$, $h_{i,l+1} = H(h_{i,l}||m_{i,l+1}||\texttt{id}||\texttt{mid})$, $m_{i,l+2} = f_{i,l+2}(m_{i,l+1})$,

$h_{i,l+2} = H(h_{i,l+1}||m_{i,l+2}||\mathtt{id}||\mathtt{mid})$ and so on. For $n > 1$, the receiver $\mathcal{R}$ computes $\mathfrak{h}$ (the root-digest of the Merkle hash tree) from the hash value $h_{i,L-1}$ and the Merkle proof for $h_{i,L-1}$. For $n = 1$, $\mathcal{R}$ sets $\mathfrak{h} = h_{1,L-1}$. $\mathcal{R}$ verifies the signature σ by checking whether

$$\Sigma.\mathsf{Verify}(pk, \mathfrak{h}, \sigma) \stackrel{?}{=} 1. \tag{2}$$

$\mathcal{R}$ outputs 1 if the equality holds; $\mathcal{R}$ outputs 0, otherwise.

Security of our EES scheme. We state and prove Theorem 1 to analyze the security of our EES scheme.

Theorem 1: Given that the hash function H is collision-resistant and the underlying digital signature scheme Σ is secure (EUF-CMA), the EES scheme described above is secure according to Definition 2 stated in Section II-C.

Proof. During the query phase of the security game, for each $1 \leq j \leq q_1$, the challenger $\mathcal{C}$ generates a signature σ_j for the data block m_j identified by $(\mathtt{id}, \mathtt{mid}_j)$ and the set of editing-functions specified by $\mathtt{spec}$, and then gives it to the PPT adversary $\mathcal{A}$. Let $\mathfrak{h}_j$ be the root-digest of the Merkle hash tree computed on the data block m_j corresponding to $\mathtt{spec}$. We can say that $\sigma_j = \Sigma.\mathsf{Sign}(sk, \mathfrak{h}_j)$ is the signature on $(m_j, \mathtt{id}, \mathtt{mid}_j, \mathtt{spec})$. We consider the following two cases.
Case I. During the challenge phase, suppose $\mathcal{A}$ selects a data block m identified by $(\mathtt{id}, \mathtt{mid})$ such that $\mathtt{mid} \neq \mathtt{mid}_j$ for all $1 \leq j \leq q_1$, and an editing-function $\mathtt{func}$ that may not be specified in $\mathtt{spec}$. Suppose $\mathcal{A}$ produces a derived data block $\overline{m}$, a verification metadata Π and a signature σ such that $\mathsf{EEVerify}$ outputs 1 for $(\overline{m}, \mathtt{mid}, \sigma, \Pi, \mathtt{func}, \mathtt{pp})$. Let $\mathfrak{h}$ be the root-digest of the Merkle tree corresponding to $(\overline{m}, \Pi)$.

We note that $\mathsf{EEVerify}$ outputs 1 only if the equality given in Eqn. 2 holds (i.e., $\Sigma.\mathsf{Verify}(pk, \mathfrak{h}, \sigma) = 1$). As $\mathsf{EEVerify}$ outputs 1 for $(\overline{m}, \Pi, \sigma)$, it follows that $\Sigma.\mathsf{Verify}(pk, \mathfrak{h}, \sigma) = 1$. Then, one of the following two cases arises:

- Case I-a: $\mathfrak{h} \neq \mathfrak{h}_j$ for all $1 \leq j \leq q_1$,
- Case I-b: $\mathfrak{h} = \mathfrak{h}_j$ for some $1 \leq j \leq q_1$.

Case I-a. We show that, if $\mathcal{A}$ can find a tuple $(\overline{m}, \Pi, \sigma)$ such that $\mathfrak{h} \neq \mathfrak{h}_j$ for all $1 \leq j \leq q_1$, then $\mathcal{A}$ can be used to break the security of the EUF-CMA signature scheme Σ. In that case, we can construct another PPT algorithm $\mathcal{B}^{\mathcal{O}_{sk}(\cdot)}$ that, given the public key pk and an access to the signing oracle $\mathcal{O}_{sk}(\cdot)$ corresponding to Σ, executes $\mathcal{A}$ as a subroutine to forge a signature. $\mathcal{B}$ simulates the challenger $\mathcal{C}$ with the following exception: $\mathcal{B}$ does not know the secret key sk, but it can query $\mathcal{O}_{sk}(\cdot)$ to get signatures on messages of its choice. Finally, $\mathcal{B}$ produces a valid forgery to Σ with the help of $\mathcal{A}$.

- Initially, $\mathcal{B}$ is given the public key. $\mathcal{B}$ decides an identifier $\mathtt{id}$ and specifies a set of editing-functions by $\mathtt{spec}$. $\mathcal{B}$ includes pk, $\mathtt{id}$, $\mathtt{spec}$ and the descriptions of H and Σ in a public parameter $\mathtt{pp}$. Finally, $\mathcal{B}$ sends $\mathtt{pp}$ to $\mathcal{A}$.
- During the j-th signing query (on the data block m_j identified by $(\mathtt{id}, \mathtt{mid}_j)$) made by $\mathcal{A}$ for $1 \leq j \leq q_1$, $\mathcal{B}$ computes $\mathfrak{h}_j$. $\mathcal{B}$ queries the signing oracle $\mathcal{O}_{sk}(\cdot)$ for a signature σ_j on $\mathfrak{h}_j$ and responds to $\mathcal{A}$'s query with σ_j.
- Let us assume that $\mathcal{A}$ finds, with probability $\epsilon_{\mathcal{A}}$, a tuple $(\overline{m}, \Pi, \sigma)$ for an editing-function metadata $\mathtt{func}$ in the challenge phase, such that $\mathfrak{h} \notin \{\mathfrak{h}_j\}_{1 \leq j \leq q_1}$ and $\Sigma.\mathsf{Verify}(pk, \mathfrak{h}, \sigma) = 1$. $\mathcal{A}$ reports $(\overline{m}, \Pi, \sigma)$ to $\mathcal{B}$ as a forgery to the EES scheme.
- $\mathcal{B}$ uses the pair $(\mathfrak{h}, \sigma)$ as a forgery to Σ.

We note that $\mathfrak{h}$ is not equal to any signing query $\mathfrak{h}_j$ made by $\mathcal{B}$ to the signing oracle $\mathcal{O}_{sk}(\cdot)$. Thus, the pair $(\mathfrak{h}, \sigma)$ is a valid forgery to the underlying EUF-CMA signature scheme Σ. The success probability of $\mathcal{B}$ is given by $\epsilon_{\mathcal{B}} = \epsilon_{\mathcal{A}}$. Given that each $\mathfrak{h}_j$ (and also $\mathfrak{h}$) can be computed in polynomial (in λ) time, the running time of $\mathcal{B}$ is also polynomial in λ (since q_1, q_2 are polynomials in λ and $\mathcal{A}$ is a PPT adversary).
Case I-b. During the challenge phase, $\mathcal{A}$ produces the pair $(\overline{m}, \Pi)$ corresponding to $\mathtt{mid}$. Let h be the hash value (computed from $(\overline{m}, \Pi)$) that corresponds to the level $l = L - 1$. That is, the root-digest $\mathfrak{h}$ of the Merkle hash tree is computed from h and its Merkle proof present in Π. Given that $\mathfrak{h} = \mathfrak{h}_j$ (for some $1 \leq j \leq q_1$) and $\mathtt{mid} \notin \{\mathtt{mid}_j\}_{1 \leq j \leq q_1}$, there exists an index j' such that $\mathfrak{h} = \mathfrak{h}_{j'}$ and $\mathtt{mid} \neq \mathtt{mid}_{j'}$. Then, $\mathcal{A}$ must have found a collision in one of the following ways:

- a collision in $\mathtt{epath}_k$ (for some $k \in [1, n]$) of $\mathcal{T}_{\mathtt{id},\mathtt{mid}_{j'},e}$ for the data block $m_{j'}$ (identified by $(\mathtt{id}, \mathtt{mid}_{j'})$), such that $h_{k,L-1}$ for $\mathcal{T}_{\mathtt{id},\mathtt{mid}_{j'},e}$ is equal to h,
- a collision in the Merkle proof for $h_{k,L-1}$ (for some $k \in [1, n]$) corresponding to the Merkle hash tree $\mathcal{T}_{\mathtt{id},\mathtt{mid}_{j'},h}$ for the data block $m_{j'}$ (in that case, $h_{k,L-1}$ for $\mathcal{T}_{\mathtt{id},\mathtt{mid}_{j'},e}$ is not equal to h, but their Merkle proofs produce the same root-digest $\mathfrak{h}_{j'} = \mathfrak{h}$).

Either of these collisions contradicts our assumption that H is collision-resistant.
Case II. During the challenge phase of the security game, suppose $\mathcal{A}$ selects a data block m_j identified by $(\mathtt{id}, \mathtt{mid}_j)$ for some $1 \leq j \leq q_1$. Suppose $\mathcal{A}$ finds two editing-functions (specified by the metadata $\mathtt{func} = (i, l)$ and $\mathtt{func}'$, respectively) such that the first one is specified in $\mathtt{spec}$, the second one is not specified in $\mathtt{spec}$ and they are not equivalent. $\mathcal{A}$ produces a derived data block $\overline{m}'$ (corresponding to $\mathtt{func}'$), a verification metadata Π and a signature σ such that the algorithm $\mathsf{EEVerify}$ outputs 1 for $(\overline{m}', \mathtt{mid}_j, \sigma, \Pi, \mathtt{func}, \mathtt{pp})$.

We note that $\mathsf{EEVerify}$ takes the value (i, l) from $\mathtt{func}$ and applies the sequence of functions $\{f_{i,l+1}, f_{i,l+2}, \ldots, f_{i,L-1}\}$ (that are available from the function tree $\mathcal{T}_{\mathtt{id},f}$ included in the public parameter $\mathtt{pp}$) on $\overline{m}'$ to obtain the subsequent outputs, if any, on $\mathtt{epath}_i$ of $\mathcal{T}_{\mathtt{id},\mathtt{mid}_j,e}$ (and the final hash value corresponding to the level $l = L - 1$). We now consider the case where the final output $\overline{m}'$ for $\mathtt{func}'$ is same as the final output $\overline{m}$ of $\mathtt{func} = (i, l)$, and the subsequent outputs lie on $\mathtt{epath}_i$. As the editing-functions are not equivalent, there exists at least one intermediate output for $\mathtt{func}'$ that is different from the corresponding intermediate output for $\mathtt{func}$ — which makes their corresponding hash values different as well. As $\mathsf{EEVerify}$ outputs 1 for $(\overline{m}', \mathtt{mid}_j, \sigma, \Pi, \mathtt{func}, \mathtt{pp})$, $\mathcal{A}$ must have found a collision in $\mathtt{epath}_i$ to reach the same hash value $h_{i,L-1}$ (that embeds i), or it must have found a collision in the Merkle hash tree for a different $h'_{i,L-1}$ such that the root-digest $\mathfrak{h}_j$ remains the same, or it must have produced a signature $\sigma \neq \sigma_j$ on a newly computed root-digest $\mathfrak{h}'_j$ — which contradicts either of our assumptions that H is collision-resistant and Σ is EUF-CMA.

For all other cases (i.e., when $\overline{m}'$ lies on $\texttt{epath}_i$ but it is not equal to $\overline{m}$, or $\overline{m}'$ lies on $\texttt{epath}_k$ for some $k \neq i$ ($1 \leq k \leq n$), or $\overline{m}'$ lies on a new path not specified by the challenger $\mathcal{C}$), if EEVerify outputs 1 for $(\overline{m}', \texttt{mid}_j, \sigma, \Pi, \texttt{func}, \texttt{pp})$, then also $\mathcal{A}$ must have found a collision in $\texttt{epath}_i$ (to get the same hash value $h_{i,L-1}$) or in the Merkle hash tree for a different $h'_{i,L-1}$ (to get the same root-digest $\mathfrak{h}_j$), or it must have produced a signature $\sigma \neq \sigma_j$ on a newly computed root-digest $\mathfrak{h}'_j$ — which contradicts either of our assumptions: H is collision-resistant, and the signature scheme Σ is EUF-CMA.

This completes the proof of Theorem 1. □

IV. Discussion

A. Efficiency of Our EES Scheme

The number of levels and the number of leaf-nodes in the function tree $\mathcal{T}_{\texttt{id},f}$ for a data block m are L and n, respectively. Let N be the number of editing-functions (i.e., the number of nodes in $\mathcal{T}_{\texttt{id},f}$) specified by the data owner $\mathcal{D}$. Thus, N is $O(n \cdot L)$. Another way of constructing $\mathcal{T}_{\texttt{id},f}$ is to represent each editing-function by a single edge from the root-node — which requires two levels in $\mathcal{T}_{\texttt{id},f}$. However, the way we have constructed $\mathcal{T}_{\texttt{id},f}$ does not require repetitive computations of same functions for a sub-path shared by different editing-functions (e.g., as shown in Figure 3, $f_{1,1} = f_{2,1} = f_{3,1} = f_{4,1}$ is computed only once). If $N \gg n$, this saves significant amount of computation for the data owner $\mathcal{D}$ (during EESign) and the editor $\mathcal{E}$ (during Process).

Given the security parameter λ, the hash values present in the editing tree and the Merkle hash tree, and the signature σ, are of size $O(\lambda)$ each. Let s_m be the size of the data block m. For simplicity in the asymptotic analysis, we assume that each derived block is also of the same size. Let block-identifiers (elements of $\mathcal{I}_B$) and message-identifiers (elements of $\mathcal{I}_M$) be represented using $O(\lambda)$ bits. Let $\mathfrak{t}_h$, $\mathfrak{t}_s$ and $\mathfrak{t}_v$ be the time required for computing a hash value (using H), signing a message and verifying a signature (using Σ), respectively.

Computational complexity. During EESign, $\mathcal{D}$ constructs the editing tree $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ and the Merkle hash tree $\mathcal{T}_{\texttt{id},\texttt{mid},h}$. Apart from computations required to obtain the derived blocks, the construction has two major computational costs: 1) computations required to obtain the hash values in $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ and $\mathcal{T}_{\texttt{id},\texttt{mid},h}$, and 2) signing the root-digest $\mathfrak{h}$ to get the signature σ. Thus, the computational overhead for $\mathcal{D}$ is $O(N \cdot \mathfrak{t}_h + n \cdot \mathfrak{t}_h + \mathfrak{t}_s)$. Similarly, the computational overhead for the editor $\mathcal{E}$ during Process is $O(N \cdot \mathfrak{t}_h + n \cdot \mathfrak{t}_h + \mathfrak{t}_v)$. We note that Edit does not incur any computational overhead for $\mathcal{E}$. On the other hand, the computational overhead for the receiver $\mathcal{R}$ during EEVerify is $O(L \cdot \mathfrak{t}_h + \log n \cdot \mathfrak{t}_h + \mathfrak{t}_v)$ — which includes computing a sequence of hashes and verifying a signature.

Storage complexity. During Process, the editor $\mathcal{E}$ constructs $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ and $\mathcal{T}_{\texttt{id},\texttt{mid},h}$, and she stores them in order to later retrieve the derived blocks and their corresponding verification metadata. Each node of $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ contains a data block and a hash value — which requires $O(s_m + \lambda)$ storage. On the other hand, each node of $\mathcal{T}_{\texttt{id},\texttt{mid},h}$ contains a hash value — which requires $O(\lambda)$ storage. Therefore, the total storage overhead for $\mathcal{E}$ is $O(N \cdot s_m + N \cdot \lambda + n \cdot \lambda)$. We note that, in our EES scheme, there is no permanent storage overhead for $\mathcal{D}$ and $\mathcal{R}$ ($\mathcal{D}$ needs only temporary storage to construct $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ and $\mathcal{T}_{\texttt{id},\texttt{mid},h}$ during EESign, but this storage is not required once $\mathcal{D}$ generates σ).

Communication complexity. After signing a data block m, the data owner $\mathcal{D}$ sends the tuple $(m, \texttt{mid}, \sigma)$ to the editor $\mathcal{E}$ — which requires $O(s_m + \lambda)$ bandwidth between $\mathcal{D}$ and $\mathcal{E}$. After editing the data block, $\mathcal{E}$ sends a tuple $(\texttt{mid}, \texttt{func}, \overline{m}, \Pi, \sigma)$ to the receiver $\mathcal{R}$ — which amounts to $O(s_m + \log n \cdot \lambda)$ bandwidth required between $\mathcal{E}$ and $\mathcal{R}$.

Trade-off between computational complexity and storage complexity. One possible way to reduce the storage overhead for $\mathcal{E}$ is as follows. After computing $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ and $\mathcal{T}_{\texttt{id},\texttt{mid},h}$, $\mathcal{E}$ deletes $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ and stores only $\mathcal{T}_{\texttt{id},\texttt{mid},h}$. However, in that case, $\mathcal{E}$ may have to compute some derived blocks (that were earlier published to a receiver) multiple times from the original data block m if $\mathcal{E}$ later wishes to publish the same derived blocks to other receivers or even to the same receiver. On the other hand, by storing $\mathcal{T}_{\texttt{id},\texttt{mid},e}$, $\mathcal{E}$ only needs to look up $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ for the derived block and its corresponding hash.

Trade-off between computational complexity and communication complexity. During Process, $\mathcal{E}$ constructs $\mathcal{T}_{\texttt{id},\texttt{mid},e}$ and $\mathcal{T}_{\texttt{id},\texttt{mid},h}$ in the same way as done by $\mathcal{D}$ during EESign. Thus, the computational overhead for $\mathcal{E}$ decreases significantly if $\mathcal{D}$ sends these trees to $\mathcal{E}$, but this increases the communication bandwidth required between $\mathcal{D}$ and $\mathcal{E}$.

B. Generalization for Multiple Data Blocks

In many applications, a data file $\mathcal{F}$ is split into multiple data blocks and different sets of editing operations are applicable on them. Our EES scheme described in Section III can be extended for multiple data blocks as follows.

Given a data file $\mathcal{F}$ split into k data blocks identified by block-identifiers $\texttt{id}_1, \texttt{id}_2, \ldots, \texttt{id}_k$ and the corresponding k sets of editing-functions $\overline{\mathrm{F}}_1, \overline{\mathrm{F}}_2, \ldots, \overline{\mathrm{F}}_k$, $\mathcal{D}$ processes each data block and the corresponding set of editing-functions separately as before and computes the root-digests $\mathfrak{h}_1, \mathfrak{h}_2, \ldots, \mathfrak{h}_k$ of the corresponding Merkle hash trees. $\mathcal{D}$ constructs another Merkle hash tree $\mathcal{T}$ over $\mathfrak{h}_1, \mathfrak{h}_2, \ldots, \mathfrak{h}_k$. Let $\mathfrak{h}$ be the root-digest of $\mathcal{T}$. $\mathcal{D}$ signs $\mathfrak{h}$ to obtain the editing-enabled signature for the data file — which requires only one signing operation for the data file $\mathcal{F}$. For a data block in this extended editing-enabled signature scheme, the size of the verification metadata Π is $O(\log n_{\max} \cdot \lambda + \log k \cdot \lambda)$ and the verification time is $O(L_{\max} \cdot \mathfrak{t}_h + \log n_{\max} \cdot \mathfrak{t}_h + \log k \cdot \mathfrak{t}_h + \mathfrak{t}_s)$, where $L_{\max}$ is the number of levels present in the function tree having the maximum number of levels, $n_{\max}$ is the number of leaf-nodes present in the function tree having the maximum number of leaf-nodes and k is the number of blocks $\mathcal{F}$ is split into.

V. Applications of Editing-Enabled Signatures

The applications of an EES scheme range from delegation of generic editing tasks to protecting privacy of sensitive data. For example, the owner $\mathcal{D}$ of an image can allow the editor $\mathcal{E}$ to put a logo in a particular location of the image (e.g., at the top left corner), or $\mathcal{D}$ can allow $\mathcal{E}$ to embed copyright information (e.g., a digital watermark) in the image before

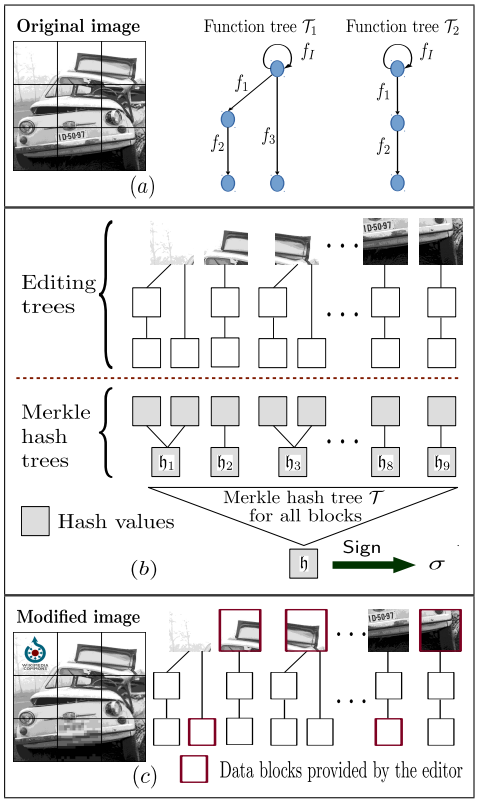


Fig. 4: An example shows how our EES scheme works for the data blocks of an image (image source: Wikimedia Commons).

publishing the image. For privacy-preserving applications, it is required that a receiver $\mathcal{R}$, given a derived data block, cannot trace back to certain data blocks which are not accessible to $\mathcal{R}$ otherwise. It is assumed that $\mathcal{E}$ and $\mathcal{R}$ do not collude with each other; otherwise, $\mathcal{R}$ can always obtain the original data from $\mathcal{E}$. We describe how to employ our EES scheme in the following privacy-preserving applications.

Privacy-preserving image/video publishing. In image processing, it is often required that sensitive parts of an image/video file must be protected from the receivers [7], [8]. For example, a surveillance camera (e.g., an IoT camera found in a car or an aerial drone) captures an image of an accident, and the owner of the camera allows a publisher (e.g., a media house) to publish the image after blurring sensitive parts of the file (e.g., human faces or number plates of cars).

Figure 4 shows how our EES scheme can be applied for privacy-preserving publication of an image. We use the extended EES scheme described in Section IV-B where the image has a grid-like structure split into nine data blocks. Figure 4(a) shows that the data owner $\mathcal{D}$ can specify different function trees for different blocks of the image. Depending on the intensity of blurring required, each block can be blurred by using the editing-function $\{f_1\}$ or $\{f_1, f_2\}$, where f_1 and f_2 are two *one-way* blurring functions. In addition, $\mathcal{D}$ specifies the editing-function $\{f_3\}$ for the $(1,1)$-th and $(1,3)$-th blocks at the top left/right corners, where f_3 puts a logo in a block. $\mathcal{D}$ specifies the function tree $\mathcal{T}_1$ for the $(1,1)$-th and $(1,3)$-th blocks and the function tree $\mathcal{T}_2$ for the rest of the blocks. Figure 4(b) shows the outlines of the editing trees and Merkle hash trees. $\mathcal{D}$ signs the final hash value $\mathfrak{h}$ to produce a signature σ. Figure 4(c) shows a modified image that $\mathcal{E}$ provides to the receiver $\mathcal{R}$, where $\mathcal{E}$ puts a logo in the $(1,1)$-th block, applies $\{f_1\}$ on the $(2,3)$-th block and applies $\{f_1, f_2\}$ on the $(3,2)$-th block (the rest of the blocks are unmodified). $\mathcal{R}$ checks the authenticity of these blocks using σ and the verification metadata. However, $\mathcal{R}$ cannot obtain any sensitive parts, including the number plate present in the $(3,2)$-th block, of the original image.

For the aforementioned example, we estimate the time required by each entity for cryptographic operations as follows. We use eBASH, a benchmarking project for hash functions [15], and eBATS, a benchmarking project for asymmetric systems [16], to estimate the time required for operations related to different hash functions and signature schemes. We use SHA-224, RSA-2048 and ECDSA over the standard NIST P-224 elliptic curve (for $\lambda = 112$) and SHA-256, RSA-3072 and ECDSA over the standard NIST P-256 elliptic curve (for $\lambda = 128$). The evaluation is done on a 4×3000MHz Intel Xeon E3-1220 processor. For different sizes of a data block, the computational overhead for the data owner $\mathcal{D}$ (to sign the image using EESign), the editor $\mathcal{E}$ (to process the image using Process) and the receiver $\mathcal{R}$ (to validate the modified image using EEVerify) are shown in Figure 5(a–c). For example, if H and Σ are realized as SHA-256 and RSA-3072 and the size of each block of the image is taken to be 256 KB, then the computational overhead due to cryptographic operations for $\mathcal{D}$, $\mathcal{E}$ and $\mathcal{R}$ are 22.09, 19.27 and 16.61 milliseconds, respectively.

Recently, Yu et al. [7] have addressed a similar problem where the owner of a video applies a blurring function f on each block of a video-frame and derives hash values computed over the blurred blocks. The owner then signs all hash values for different frames to get the final signature on the video. This scheme can be realized using our extended EES scheme, where the function tree for a block consists of a root-node and a leaf-node ($n = 1, L = 2$) with the edge between them representing f. We note that our EES scheme is more powerful as $\mathcal{D}$ can combine different blurring functions to get different levels of blurring. Moreover, privacy-preserving operations can be coupled with generic editing tasks — that enables $\mathcal{D}$ to choose a wide range of editing-functions for her data.

Privacy-preserving data publishing. In recent years, privacy-preserving data publishing (PPDP [17], [18]) has earned substantial attention from the research community. PPDP allows data to be published in a way that sensitive information of an individual record owner is still protected. k-anonymity [19] is a popular PPDP technique used to anonymize a table T, such that each group of records sharing a common value for a set of attributes has cardinality (at least) k. That is, each record is indistinguishable from at least $k-1$ other records sharing a common value for a set of attributes. It helps in protecting privacy even when certain public (external) information on the record owners is available. We illustrate the idea using Table I. Suppose a hospital wants a research center to publish (apparently anonymous) patient-records shown in Table I(a). We assume that: Table I(b) is an external table that a receiver of the published records has access to, and it is known

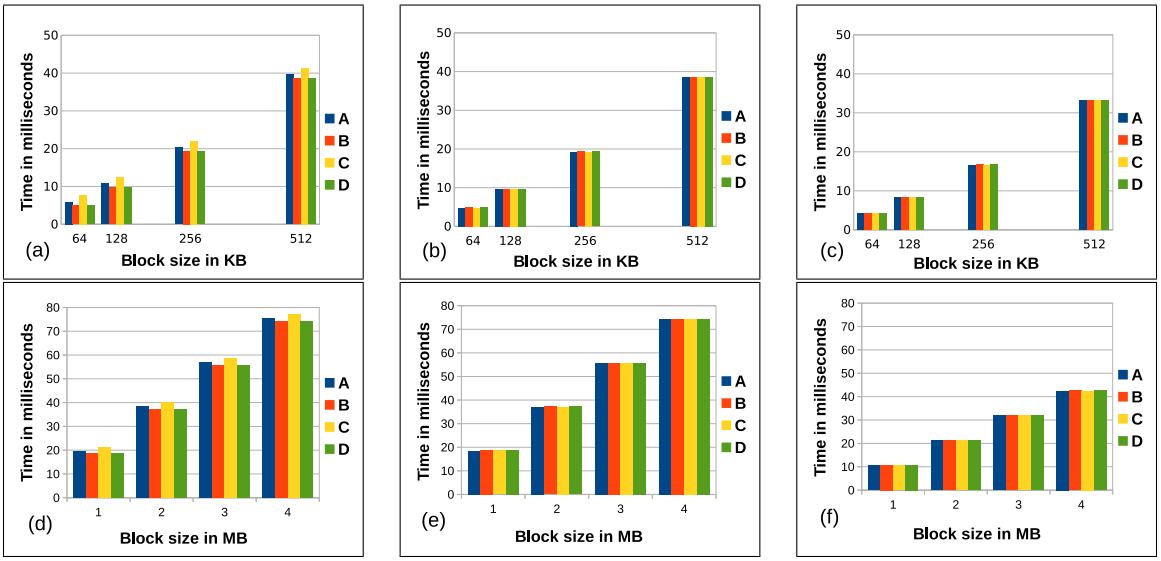


Fig. 5: Time required for performing cryptographic operations by $\mathcal{D}$ (during EESign), $\mathcal{E}$ (during Process) and $\mathcal{R}$ (during EEVerify) in the examples: (a–c) *privacy-preserving image/video publishing*, (d–f) *privacy-preserving data publishing*. A: SHA-224 + RSA-2048; B: SHA-224 + ECDSA over NIST P-224 elliptic curve; C: SHA-256 + RSA-3072; D: SHA-256 + ECDSA over NIST P-256 elliptic curve.

TABLE I: An example of privacy-preserving data publishing

(a) Patient table

Job	Sex	Age	Disease
Engineer	Male	42	Hepatitis
Engineer	Male	41	Hepatitis
Lawyer	Male	44	HIV
Writer	Female	36	Flu
Writer	Female	38	HIV
Dancer	Female	30	HIV
Dancer	Female	33	Hepatitis

(b) External table

Name	Job	Sex	Age
Alice	Writer	Female	36
Bob	Engineer	Male	41
Cathy	Writer	Female	38
Doug	Lawyer	Male	44
Emily	Dancer	Female	30
Fred	Engineer	Male	42
Gladys	Dancer	Female	33
Henry	Lawyer	Male	39
Irene	Dancer	Female	32

(c) 2-anonymous patient table

Job	Sex	Age	Disease
Professional	Male	$[40-45)$	Hepatitis
Professional	Male	$[40-45)$	Hepatitis
Professional	Male	$[40-45)$	HIV
Artist	Female	$[35-40)$	Flu
Artist	Female	$[35-40)$	HIV
Artist	Female	$[30-35)$	HIV
Artist	Female	$[30-35)$	Hepatitis

that a person, having a record in Table I(a), has a record in Table I(b). Based on {Job, Sex, Age}, the receiver can identify, with certainty, some patients suffering from specific diseases.

Figure 6 shows an example of how our extended EES scheme allows the hospital to sign Table I(a) in such a way that the research center can later perform different anonymization operations on the data. The hospital specifies three (*many-to-one*) generalization functions f_1, f_2 and f_3 for data blocks (i.e., columns of the table). The function f_1 applicable on the first block classifies each record r in the block as follows: Engineer/Lawyer is classified as Professional, and Writer/Dancer is classified as Artist. The functions f_2 and f_3 applicable on the third block classify each record r in the block using $\left(\lfloor \frac{r}{5} \rfloor \cdot 5, \lfloor \frac{r}{5} \rfloor \cdot 5 + 5\right)$ and $\left(\lfloor \frac{r}{10} \rfloor \cdot 10, \lfloor \frac{r}{10} \rfloor \cdot 10 + 10\right)$, respectively. Figure 6(a) shows the function trees for different blocks

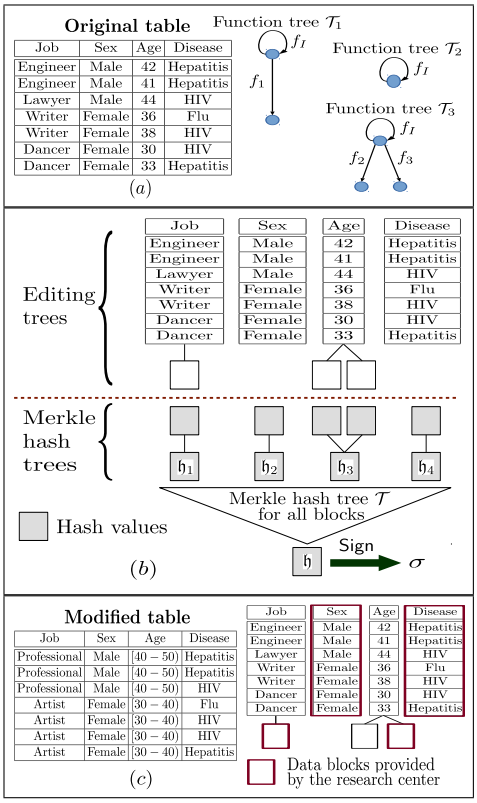


Fig. 6: An example shows how our EES scheme works for different data blocks of Table I(a).

of the table ($\mathcal{T}_1$ for the first block, $\mathcal{T}_2$ for the second and fourth blocks, and $\mathcal{T}_3$ for the third block). Figure 6(b) shows the outlines of the editing trees and Merkle hash trees. The hospital signs the final hash value $\mathfrak{h}$ to get σ. The research center later applies f_1 on the first block and f_3 on the third

block to get the *3-anonymous* table shown in Figure 6(c). If the research center applies f_2 on the third block and f_1 on the first block, the modified table is *2-anonymous*. Thus, given different generalization functions, the research center can use any of them to achieve the desired level of anonymity. The receiver cannot obtain certain sensitive information about individual patients that she could have obtained from Table I(a).

For the example given in Figure 6, we estimate the time required by $\mathcal{D}$, $\mathcal{E}$ and $\mathcal{R}$ using the same cryptographic primitives and benchmarks as used in the previous example. Let us assume that the size of each column (that is considered to be a data block) of Table I(a) is same. For different sizes of a data block, the computational overhead for $\mathcal{D}$ (to sign the table using EESign), $\mathcal{E}$ (to process the table using Process) and $\mathcal{R}$ (to validate the modified table using EEVerify) are shown in Figure 5(d–f). For example, if H and Σ are realized as SHA-256 and RSA-3072 and the size of each column of the table is taken to be 3 MB, then the computational overhead due to cryptographic operations for $\mathcal{D}$, $\mathcal{E}$ and $\mathcal{R}$ are 58.57, 55.74 and 31.86 milliseconds, respectively.

VI. Related Work

In this section, we discuss some other kinds of signature schemes that allow one to modify signed messages in certain ways, and we highlight how they are different from our editing-enabled signatures.

A similar line of research includes sanitizable signatures [9], [10], [20], [21], [22], [23]. The notion of sanitizable signatures was introduced by Ateniese et al. [9], where the signer of a message authorizes a set of designated parties, named sanitizers, to modify respective parts of the signed message without changing the original signature. The sanitizers must be fixed and known to the signer since their public keys are used in the signing process. The underlying idea is as follows. The signer applies a chameleon hash function [24], which is a trapdoor hash function, on a message and signs the output hash using a digital signature scheme. The hash function uses the public key of a sanitizer such that the sanitizer can later modify *arbitrarily* the corresponding part of the message specified by the data owner without changing the hash value, thus keeping the signature still valid. In order to find a collision to the hash function corresponding to the modified message, the sanitizer uses her secret key. Finally, the recipient can validate the signature on the modified message since it produces the same hash value as the original message. In comparison, the data owner in an EES scheme uses only her signing key to sign the data based on the set of editing-functions, and the data owner can thus choose any editor after she signs the data. This is more practical since in many applications where the data is produced and signed, it is impossible for data owners to know who the editors will be. Moreover, editing-enabled signatures provide more control over the data to the data owner in that the data owner solely decides the set of editing-functions applicable for the data and the editor *cannot* generate a valid signature for arbitrarily modified data.

A redactable signature is a type of homomorphic signatures [11], [25]. Redactable signatures [11], [12], [26] consider *only* deletions in different locations of a data file. The signer signs the data file such that each location can be later replaced by a special symbol in order to denote a deletion. The signature on the redacted data file can be obtained from the original signature. Content-extraction signatures [27] are redactable signatures where the redaction operation aims to remove XML nodes. Unlike these signatures, the range of operations applicable in an EES scheme is vast rather than being limited to deleting (or removing) some parts of the data.

We only mention some other homomorphic signatures that include transitive signatures [28], [29] and homomorphic signatures for network coding [30]. However, unlike EES schemes, these homomorphic signature schemes are designed for *specific* modification operations only (e.g., in a transitive signature scheme, given the signatures for two edges (v_i, v_j) and (v_j, v_k) of a graph, one can produce a signature on the edge (v_i, v_k)).

VII. Conclusion

In this work, we have introduced the notion of editing-enabled signatures that provide flexibility to a data owner, while specifying a set of modification operations for the data, and restrict an arbitrary editor to modify the data with a specified operation and authenticate the modified data to a receiver. The editor does not need to interact with the data owner for such modifications. As the signing algorithm does not require the credentials of an editor, the data owner can later send the authenticated data to any editor of her choice. We note that the existing signature schemes aiming to achieve similar goals either serve for a limited number of modification operations or cannot refrain the editor from modifying the data arbitrarily. Following the notion of editing-enabled signatures, we have constructed an editing-enabled signature (EES) scheme based on mostly lightweight cryptographic primitives (viz., collision-resistant hash-chains) — which makes it suitable for many applications involving resource-constrained data owners. We have formally defined the security of editing-enabled signatures and provided the security analysis of our EES scheme. We have also analyzed the efficiency of our EES scheme based on different complexity parameters. Finally, we have demonstrated how our EES scheme can be efficiently used for authentication in two applications – privacy-preserving image/video publishing and privacy-preserving data publishing.

References

[1] M. Weiser, "Ubiquitous computing," *Computer*, vol. 26, no. 10, pp. 71–72, 1993.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] A. Shamir, "Financial cryptography: Past, present, and future (anniversary keynote)," February 2016, International Conference on Financial Cryptography and Data Security.

[4] M. Kuzin, Y. Shmelev, and V. Kuskov, "New trends in the world of IoT threats," September 2018, https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/.

[5] W. McGrath, M. Etemadi, S. Roy, and B. Hartmann, "fabryq: Using phones as gateways to prototype internet of things applications using web scripting," in *ACM Symposium on Engineering Interactive Computing Systems*, 2015, pp. 164–173.

[6] Microsoft, "Microsoft Azure IoT reference architecture (version 2.1)," September 2018, https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-introduction.

[7] H. Yu, J. Lim, K. Kim, and S. Lee, "Pinto: Enabling video privacy for commodity IoT cameras," in *ACM Conference on Computer and Communications Security*, 2018, pp. 1089–1101.

[8] J. Obermaier and M. Hutle, "Analyzing the security and privacy of cloud-based video surveillance systems," in *ACM International Workshop on IoT Privacy, Trust, and Security*, 2016, pp. 22–28.

[9] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik, "Sanitizable signatures," in *European Symposium on Research in Computer Security*, 2005, pp. 159–177.

[10] K. Miyazaki, G. Hanaoka, and H. Imai, "Digitally signed document sanitizing scheme based on bilinear maps," in *ACM Symposium on Information, Computer and Communications Security*, 2006, pp. 343–354.

[11] R. Johnson, D. Molnar, D. X. Song, and D. A. Wagner, "Homomorphic signature schemes," in *The Cryptographer's Track at the RSA Conference*, 2002, pp. 244–262.

[12] E. Chang, C. L. Lim, and J. Xu, "Short redactable signatures using random trees," in *The Cryptographer's Track at the RSA Conference*, 2009, pp. 133–147.

[13] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology - CRYPTO*, 1987, pp. 369–378.

[14] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, April 1988.

[15] D. J. Bernstein and T. Lange, "eBASH: ECRYPT benchmarking of all submitted hashes," July, 2016, http://bench.cr.yp.to/ebash.html.

[16] ——, "eBATS: ECRYPT benchmarking of asymmetric systems," August, 2018, https://bench.cr.yp.to/ebats.html.

[17] J. Wang and W. Lin, "Privacy preserving anonymity for periodical SRS data publishing," in *IEEE International Conference on Data Engineering*, 2017, pp. 1344–1355.

[18] J. Qian, F. Han, J. Hou, C. Zhang, Y. Wang, and X. Li, "Towards privacy-preserving speech data publishing," in *IEEE Conference on Computer Communications*, 2018, pp. 1079–1087.

[19] P. Samarati and L. Sweeney, "Generalizing data to provide anonymity when disclosing information (abstract)," in *ACM Symposium on Principles of Database Systems*, 1998, p. 188.

[20] R. W. F. Lai, T. Zhang, S. S. M. Chow, and D. Schröder, "Efficient sanitizable signatures without random oracles," in *European Symposium on Research in Computer Security*, 2016, pp. 363–380.

[21] C. Brzuska, H. C. Pöhls, and K. Samelin, "Efficient and perfectly unlinkable sanitizable signatures without group signatures," in *European Workshop on Public Key Infrastructures, Services and Applications*, 2013, pp. 12–30.

[22] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures," in *International Conference on Practice and Theory in Public-Key Cryptography*, 2017, pp. 152–182.

[23] M. Fischlin and P. Harasser, "Invisible sanitizable signatures and public-key encryption are equivalent," in *International Conference on Applied Cryptography and Network Security*, 2018, pp. 202–220.

[24] H. Krawczyk and T. Rabin, "Chameleon signatures," in *Network and Distributed System Security Symposium*, 2000.

[25] D. Catalano, D. Fiore, and L. Nizzardo, "On the security notions for homomorphic signatures," in *International Conference on Applied Cryptography and Network Security*, 2018, pp. 183–201.

[26] C. Brzuska, H. Busch, Ö. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder, "Redactable signatures for tree-structured data: Definitions and constructions," in *International Conference on Applied Cryptography and Network Security*, 2010, pp. 87–104.

[27] R. Steinfeld, L. Bull, and Y. Zheng, "Content extraction signatures," in *International Conference on Information Security and Cryptology*, 2001, pp. 285–304.

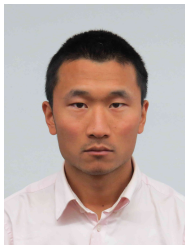
[28] S. Micali and R. L. Rivest, "Transitive signature schemes," in *The Cryptographer's Track at the RSA Conference*, 2002, pp. 236–243.

[29] M. Bellare and G. Neven, "Transitive signatures based on factoring and RSA," in *ASIACRYPT*, 2002, pp. 397–414.

[30] D. Catalano, D. Fiore, and B. Warinschi, "Efficient network coding signatures in the standard model," in *International Conference on Practice and Theory in Public-Key Cryptography*, 2012, pp. 680–696.

BINANDA SENGUPTA received his PhD degree in Computer Science from Indian Statistical Institute, India. His broad research area includes applied cryptography, cloud computing, security and privacy. He is currently a postdoctoral research fellow affiliated with the School of Information Systems, Singapore Management University.

YINGJIU LI is currently a Ripple Professor in the Computer and Information Science Department at the University of Oregon. His research interests include IoT Security and Privacy, Mobile and System Security, Applied Cryptography and Cloud Security, and Data Application Security and Privacy. He has published over 140 technical papers in international conferences and journals, and served in the program committees for over 80 international conferences and workshops, including top-tier cybersecurity conferences and journals.



YANGGUANG TIAN received his Ph.D degree in Applied Cryptography from University of Wollongong, Australia. He is a research fellow at the School of Information Systems, Singapore Management University. His research interests include user authentication, privacy protection, applied cryptography and network security.

ROBERT H. DENG is AXA Chair Professor of Cybersecurity and Director of the Secure Mobile Centre, School of Information Systems, Singapore Management University. His research interests are in the areas of data security and privacy, cloud security and Internet of Things security. He is an IEEE Fellow.