This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

Immutable Authentication and Integrity Schemes for Outsourced Databases

Attila A. Yavuz, *Member, IEEE*

Abstract—Database outsourcing enables organizations to offload their data management overhead to the external service providers. Immutable signatures are ideal tools to provide authentication and integrity for such applications with an important property called immutability. Signature immutability ensures that, no attacker can derive a valid signature for unposed queries from previous queries and their corresponding signatures. This prevents an attacker from creating his own de-facto services via such derived signatures. Unfortunately, existing immutable signatures are very computation/communication costly, which make them impractical for real-life applications.

In this paper, we developed three new schemes called *Practical and Immutable Signature Bouquets* ($PISB$), which achieve efficient immutability for outsourced databases. $PISB$ schemes are simple, non-interactive, and computation/communication efficient. Our generic scheme can be constructed from any aggregate signature coupled with a standard signature. Our specific scheme is constructed from Condensed-RSA and Sequential Aggregate RSA. It has a low verifier computational overhead and compact signature. Our third scheme offers the lowest end-to-end delay among existing alternatives by enabling efficient signature pre-computability. We provide formal security analysis of $PISB$ schemes (in Random Oracle Model) and give a theoretical analysis on the relationship between signature immutability and signature extraction. We also showed that $PISB$ schemes are more efficient than previous alternatives.

Index Terms—Applied cryptography; outsourced databases; immutable digital signatures; distributed systems.

1 Introduction

It is a growing trend that the data is outsourced and being managed on remote servers, which are maintained by third party outsourcing vendors. One such data outsourcing approach is "database as a service" (DAS) model [1], in which clients outsource their data to a database service provider[1,2] that offers a reliable maintenance/access for the hosted data [2].

Data outsourcing can significantly reduce the cost of data management (e.g., via continuous service, expertise, maintenance) and therefore it is highly beneficial for entities with limited management capabilities such as small to medium businesses [2]–[4]. However, despite its merits, data outsourcing brings various security challenges, since the sensitive data is hosted in a (semi) untrusted environment. These security challenges include but not limited to the confidentiality [5], access privacy [6], authentication and integrity [7]. Another challenge is to provide the security efficiently such that the data outsourcing still remains practical and cost efficient.

The focus of this paper is to provide authentication and integrity of outsourced data via aggregate signatures (e.g., [8]), while also guaranteeing a vital security property called *signature immutability* in a *practical* manner.

Differences between this article and its preliminary version in [9]: In this article, we develop a new construction and also give a more comprehensive security and performance analysis over the preliminary version in [9]: (i) We introduce a new scheme called $PISB\text{-}RP$ that offers the lowest end-to-end delay among existing alternatives. (ii) We investigate the relationship between signature immutability and aggregate signature extraction [8], [10], which has been omitted in previous outsourced database authentication schemes (e.g., [3], [7]). We proved that the signature extraction is a necessary condition for some immutable signature constructions such as $PISB\text{-}RP$. (iii) We discuss pros and cons of various $PISB$ instantiations by highlighting their performance characteristics.

Attila A. Yavuz is with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 USA, (e-mail: attila.yavuz@oregonstate.edu)

1. http://www.ibm.com/software/data/db2

2. http://www-935.ibm.com/services/us/en/it-services/storage-and-data-services.html

1.1 System and Data Model

We follow Mykletun et al.'s Outsourced Database Model (ODB) [3], [7] as a variant of "database as a service" [1].

System Model: There are three types of entities in the system; data owners, server (database service provider) and data queriers (clients). These entities behave as follows.

• *Data Owners:* A data owner can be a single or a logical entity such as an organization. Each data owner in the system signs her database elements (e.g., each tuple separately) and then outsources them along with their signatures to *the server*. This protects the integrity and authentication of outsourced data against *both* the server and outside adversaries (e.g., in the case of the server is compromised).

The data owner computes the individual signature of each database element (e.g., each tuple) with an aggregate signature scheme (e.g., [8]), which allows the combination of these signatures according to the content of a query. This enables the server to reply any query on the outsourced data with a *compact constant size signature* (instead of sending a signature for each element in the query, which entails a linear communication overhead). This outsourcing step is performed *offline*, and therefore its cost is not the main concern.

• *Server (Service Provider):* The server maintains the data and handles the queries of *data queriers*. The server is trusted

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

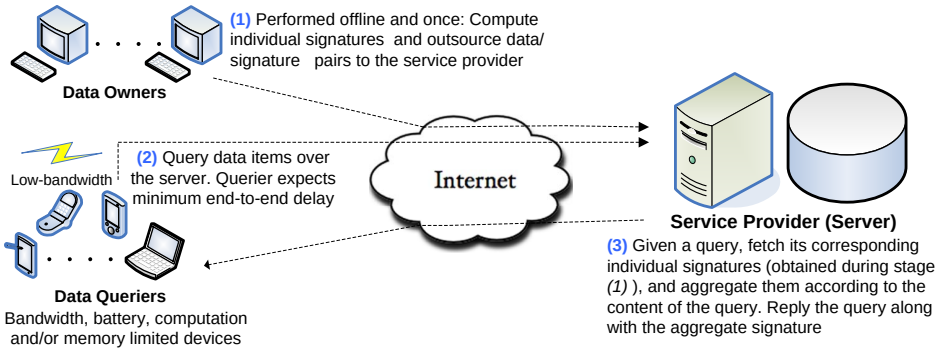


Fig. 1: Mykletun et al.'s Outsourced Database Model (ODB)

with these services, but it is *not* trusted with the integrity and authentication of the data. Hence, each data owner digitally signs her data before outsourcing it as described previously.

Once a *data querier* (i.e., clients who perform data queries) queries the server, the server computes a constant size signature by aggregating the corresponding individual signatures of database elements associated with this query. Recall that the server knows these individual signatures, since the data owner provided all individual signatures to the server at the offline phase. The server then performs necessary cryptographic operations to ensure the immutability of this aggregate signature. Observe that the server faithfully follows the immutability operations, since the immutability prevents external parties to offer similar services free of charge.

The query handling phase is performed online. The server is expected to handle larger number of queries simultaneously with a minimum end-to-end delay. Therefore, the cost of signature immutability operations are *highly critical*.

• *Data Queriers (Clients)*: Queriers are heterogeneous entities, which may be resource-constrained in terms bandwidth, battery and/or computation (e.g., a PDA). A querier can make a query on the database elements belonging to a single or multiple data owners. The former is called *single signer queries* while the latter is called *multiple signer queries*. The data querier verifies the aggregate signature of her query, along with cryptographic tokens transmitted for the immutability.

Figure 1 summarizes the ODB model described above.

Data Model: We assume that the data is managed with a traditional relational database management system and the queries are formulated with SQL. Our work handles only SQL queries involving SELECT clauses, which return the selection of a set of records or fields matching a given predicate.

The granularity of data integrity and authentication may vary according to the application (e.g., attribute level). For example, one possible choice is to provide them at the tuple level (i.e., sign each tuple individually), which offers a balance between the storage, transmission and computation overheads introduced by the cryptographic scheme [3].

1.2 Problem Statement: Signature Mutability in ODBs

Ability to aggregate different signatures into a single one is advantageous, as exemplified in the above ODB model. However, this property may have undesirable security implications.

Remark that aggregate signatures exhibit homomorphic properties, and therefore are malleable by design [11]. That is, any party can derive valid aggregate signatures, without explicitly querying them, by just combining aggregate signatures of previously queried messages. For instance, let $ASig$ be a multiplicative aggregate signature scheme (e.g., C-RSA as in Definition 5). In the above ODB setting, assume that the data owner provides an aggregate signature σ on items $m_1, \ldots, m_k$ to the server (e.g., a music album comprised of different songs). Later, the data owner issues an another signature σ' on items $m_{k+1}, \ldots, m_l$. Notice that any querier (e.g., the client) can derive a valid signature on query elements $m_1, \ldots, m_l$ (that have *not* been queried before) by simply computing $\overline{\sigma} = \sigma \cdot \sigma'$. As an example, this signature may permit the querier to sell and re-distribute two separate albums together without obtaining any authorization from the data owner.

This property of the aggregate signatures is called as *signature mutability* and has several undesirable effects on real-life applications. Another example is *content access control mechanisms* for outsourced databases. Assume that the data owner requires the server to enforce an access control policy, in which each client can access only certain parts of the database via an access token (i.e., a signature). Each client may possess different access privileges. However, if clients collude then they can exploit signature mutability to derive an access token (a mutated aggregate signature), which provides an access right to them beyond their actual privileges.

Intuitively, the *signature immutability* refers to the difficulty of computing new valid aggregated signatures from a set of other aggregated signatures [7]. The term "immutable signature bouquets" [7], [12] refers a set (bouquets) of aggregate signatures, which cannot be mutated by an adversary but only can be aggregated by the permitted entity (i.e., the server in our case). We give further details about the existing immutable signatures in Section 1.3, and provide formal definitions for signature immutability in Section 3.

1.3 Related Work and Limitations

Aggregate signatures aggregate n individual signatures associated with n different users (or data items) into a single, compact signature. The first aggregate signature scheme was proposed in [8], and then several new schemes achieving more advanced properties were developed (e.g., sequentiality [13], ID-based for low storage overhead [14]). As discussed in Section 1.2, aggregate signatures are mutable by definition, which creates security problems for some applications. Achieving efficient and practical aggregate signature immutability, especially in the context of data outsourcing model described in Section 1.1, is the main objective of this paper.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

We give the details of aggregate, sequential aggregate [15], [16] and condensed signatures [3], [7] in Section 2.

Mykletun et al. [7] introduce signature immutability techniques to address the signature mutability problem described in Section 1.2. These immutable signatures are called as "Immutable Signature Bouquets", which refer to a set (bouquets) of aggregate signatures that can be aggregated by the permitted entity (i.e., the server in our ODB model) but cannot be used to derive other valid aggregate signatures (i.e., the signature bouquets are immutable). Their RSA-based techniques prevent an adversary from deriving new signatures by hiding the actual aggregate signature via an interactive Guillou-Quisquater (GQ) [17] based protocol. This approach is interactive and therefore introduces high communication overhead and end-to-end delay. Their non-interactive RSA variant uses a signatures of knowledge method, which substantially increases the computational cost and has large signature size. Their BGLS signature method iBGLS [8] offers a small signature size, but it is very computationally costly due to cryptographic pairing operations. Hence, none of these techniques are suitable for nowadays task-intensive and heterogeneous outsourcing applications. Notice that these are the only general purpose immutable signature constructions (to the best of our knowledge) and therefore are the main counterpart of our schemes. We provide an extensive comparison of these schemes with our proposed $PISB$ schemes in Section 6.

Immutable signatures serve as a building block for various data outsourcing applications such as database-as-a-service [12] and data protection methods (e.g., [18], [19]). They are also used with other cryptographic primitives such as forward-secure signatures to obtain secure audit logging systems (e.g., [20]–[22]). However, immutability techniques used in these secure logging systems require linear overhead and therefore are not suitable for our envisioned applications.

Offline/online signatures (e.g., [23]) and some special pre-computation techniques (e.g., [24]) are also related to our constructions. In offline/online signatures (e.g., [23], [25], [26]), the signer prepares a token during the offline phase, which can be used to sign any message during the online phase without performing expensive operations (e.g., modular exponentiation). Offline/online signatures (e.g., [23], [25], [26]) offer generic signature pre-computation. That is, any signature scheme can be executed in offline/online mode. However, majority of these methods rely on one-time signatures to achieve online computational efficiency generically. Hence, for each signature, a one-time public key and signature must also be generated and transmitted. The size of these public keys and signatures are very large (e.g., 3-5 KB for HORS [27] and similar overheads for its variants such as [28]–[31]). Hence, despite their computational benefits, these methods are not ideal to accelerate immutable database authentication methods. Some specific pre-computation methods such as pre-computed DSA tokens [24] can be considered as a special instantiation of offline/online signatures. Different pre-computation methods for Discrete Logarithm Problem (DLP) based signatures have also been developed in [32] (e.g., for Schnorr signatures [33] but also applicable to several other Meta-Elgamal signatures [34]). Pre-computation methods for RSA-type signatures also have been proposed in [35], but they can only work on strictly pre-structured messages (e.g., as in some command and control protocols). Hence, they are not applicable for our envisioned applications.

Another related work is on the Authenticated Data Structures (ADS) [36], [37]. An ADS is a method for data authentication, in which the (untrusted) server (i.e., prover) answers questions of a querier (i.e., verifier) on the data structure and provides extra information used to generate a proof that the answers are valid [38]. RSA-accumulator [39] based ADSs have been proposed in [40], [41]. Merkle-hash tree [42] is used to construct several ADSs. Verifiable B-trees (i.e., a B-tree using Merkle-hash tree) in [43] and MB-trees (a VB-tree with light hash function instead of heavy signatures) in [44] are some notable Merkle-hash tree based ADS variants. Another line of ADSs is based on authenticated skip-lists [45], from which several constructions have been developed (e.g., [46], [47]). Kupcu et. al. in [38] introduced a Hierarchical ADS (HADS), which offer compact proof sizes for multi-clause queries. ADSs are also used for secure logging purposes in different settings as in [48].

ADSs generally use a proper digital signature scheme as a building block, where the prover returns a proof (i.e., a signature) for per-item in the answer. Remark that, as mentioned in [12], [41], [48], digital signatures with additional capabilities (e.g., in our case immutable aggregation) can improve the efficiency of ADSs by serving as a special building block. Assume that the client and server in our data model act as the prover and verifier, respectively, for an ADS. In this case, $PISB$ schemes can be used to generate a small-constant size and non-malleable (i.e., immutable) proof to offer compactness. That is, instead of returning a proof for per-item in a query with l-items, one can utilize $PISB$ schemes to create a single-compact proof on l-items as an aggregate signature, which cannot be used to alter any other proofs thanks to the immutability property. For instance, our RSA-based schemes (e.g., $PISB\text{-}CSA\text{-}RSA$) can serve as a compact and non-malleable building block for RSA-accumulator based ADSs (e.g., [40], [41]).

Note that our work focuses on the authentication and integrity services. There are extensive studies on the data privacy for outsourced database systems (e.g., [49]), which are complementary to our work.

1.4 Our Contribution

To address the limitations of existing immutable aggregate signature constructions, we develop cryptographic schemes called *Practical Immutable Signature Bouquets (PISB)*, which is suitable for outsourced database systems. Specifically, we developed three $PISB$ schemes: *(i)* Condensed-RSA (C-RSA) and Sequential Aggregate RSA (SA-RSA) based scheme called $PISB\text{-}CSA\text{-}RSA$, *(ii)* a generic scheme called $PISB\text{-}Generic$, and *(iii)* a scheme that enables efficient immutable aggregate signature pre-computation called $PISB\text{-}RP$. We summarize the desirable properties of our schemes below:

1. *Non-interactive Signature Immutability*: $PISB$ schemes do not require any multi-round interaction among the server and queriers. Hence, they are much more communication efficient than previous alternatives. For instance, our $PISB\text{-}CSA\text{-}RSA$ incurs only 1KB communication overhead, while GQ-based scheme in [3], [7] requires 9KB. Moreover, the non-interactive nature of our schemes make them packet loss tolerant, which is a desirable property for mobile and ad-hoc clients (queriers).
2. *High Computational Efficiency and Pre-computability*: $PISB$ schemes are much more computationally efficient than their counterparts.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

- $PISB$-CSA-RSA is a client efficient scheme being a magnitude of time faster than SKROOT-based and iBGLS schemes in [3], [7]. Therefore, $PISB$-CSA-RSA is an ideal alternative for battery and/or computational limited clients such as mobile and hand-held devices. It is also plausibly efficient at the server side while achieving this client efficiency.
- $PISB$ schemes are the only alternatives that enable *efficient immutable signature pre-computation*. $PISB$-$Generic$ permits specific pre-computation methods such as ECDSA tokens [24] to be implemented. $PISB$-RP offers pre-computable immutable signatures with the lowest computational and communication overhead among all existing alternatives.

3. *Communication Efficiency*: $PISB$ schemes offer smaller signature sizes compared to other alternatives:
   - $PISB$-CSA-RSA and $PISB$-RP (with C-RSA) are the only RSA-based schemes that can compute a compact immutable aggregate signature, which makes them more communication efficient than their counterparts [3], [7]. $PISB$-$Generic$ has a much smaller signature size than RSA-based schemes and also has a comparable signature size with iBGLS in [3], [7] (while being much more computationally efficient).
   - $PISB$-RP introduces only κ-bit (e.g., 80-bit) transmission overhead over $PISB$-CSA-RSA while enabling signature pre-computability. Hence, $PISB$-RP is much more communication and storage efficient than other alternatives with pre-computability such as offline/online signatures (e.g., [25], [26]) and $PISB$-$Generic$ instantiations (see Section 6).
4. *Formal Security Analysis and Provable Security*: Previous works (e.g., [3], [7]) give only heuristic security arguments regarding the signature immutability. Our work is the only one providing a formal security model and proofs for the signature immutability (in Random Oracle Model (ROM) [50]). We also highlight the relationship between aggregate signature extraction (see Section 3.2) and signature immutability, which has been omitted in previous outsourced database authentication schemes.
5. *Low Delay*: High computational/communication efficiency and non-interactive nature of $PISB$ schemes enable a low end-to-end delay. $PISB$-RP with C-RSA offers *two magnitudes of times lower end-to-end delay* than that of schemes proposed in [3], [7].

Limitations: We highlight some limitations of $PISB$ as follows: *(i)* Any $PISB$ instantiation (e.g., $PISB$-$Generic$ and $PISB$-RP with BGLS [8]) with multiple signer type aggregate signature can handle queries from multiple data owners. However, despite being the fastest $PISB$ instantiation, $PISB$-CSA-RSA can only handle single data owner queries. This may pose a limitation for certain applications enforcing queries with multiple data owners. *(ii)* Similar to its counterparts [3], [7], $PISB$ schemes can only handle SELECT, but not AVG type of queries (e.g., $PISB$ cannot return an aggregate signature on a sum of signed messages).

2 Preliminaries

In this section, we give the notation and preliminary definitions used by our schemes.

Notation: Operators $||$ and $|x|$ denote the concatenation operation and the bit length of variable x, respectively. $x \xleftarrow{\$} \mathcal{S}$ denotes that variable x is randomly and uniformly selected from set $\mathcal{S}$. $|\mathcal{S}|$ denotes the cardinality of set $\mathcal{S}$. $\{x_i\}_{i=0}^{l}$ denotes $(x_0, \ldots, x_l)$. We denote by $\{0,1\}^*$ the set of binary strings of any finite length.

Definition 1 A signature scheme Sig is a tuple of three algorithms $(Kg, Sign, Ver)$ defined as follows:
- $(sk, pk) \leftarrow Sig.Kg(1^\kappa)$: Given the security parameter 1^κ, the key generation algorithm returns a private/public key pair (sk, pk) as the output.
- $s \leftarrow Sig.Sign(sk, m)$: The signing algorithm takes sk and a message m as the input. It returns a signature s as the output.
- $c \leftarrow Sig.Ver(pk, m, s)$: The signature verification algorithm takes pk, m and s as the input. It outputs a bit c, with $c = 1$ meaning *valid* and $c = 0$ meaning *invalid*.

The standard security notion for a signature scheme is *Existential Unforgeability under Chosen Message Attacks (*EU-CMA*)* [51], which is defined below.

Definition 2 EU-CMA experiment for Sig is defined as follows:
- *Setup.* Challenger algorithm $\mathcal{B}$ runs the key generation algorithm as $(sk, pk) \leftarrow Sig.Kg(1^\kappa)$ and provides pk to the adversary $\mathcal{A}$.
- *Queries.* Beginning from $j = 1$ and proceeding adaptively, $\mathcal{A}$ queries $\mathcal{B}$ on any message m_j of her choice up to q_s messages. For each query j, $\mathcal{B}$ computes $s_j \leftarrow Sig.Sign(sk, m_j)$ as the signing oracle of $\mathcal{A}$ and returns s_j to $\mathcal{A}$.
- *Forgery.* Finally, $\mathcal{A}$ outputs a forgery (m^*, s^*) and wins the EU-CMA experiment, if $Sig.Ver(pk, m^*, s^*) = 1$ and m^* was not queried to $\mathcal{B}$.

Sig is (t, q_s, ϵ)-EU-CMA secure, if no $\mathcal{A}$ in time t making at most q_s signature queries has an advantage at least with probability ϵ in the above experiment.

An aggregate signature scheme (e.g., [8]) aggregates multiple signatures of different signers into a single compact signature. Hence, it can be used for *multiple querier* applications.

Definition 3 An aggregate signature scheme $ASig$ is a tuple of four algorithms $(Kg, Sign, Agg, Ver)$ defined as follows:
- $(\overrightarrow{sk}, \overrightarrow{pk}) \leftarrow ASig.Kg(1^\kappa)$: Given the security parameter 1^κ and a set of signers $\mathbb{U} = \{1, \ldots, u\}$, the aggregate key generation algorithm generates a private/public key pair (sk_i, pk_i) for $i = 1, \ldots, u$, as in Definition 1 key generation algorithm. The aggregate key generation algorithm returns a private/public key pair $\overrightarrow{sk} = (sk_1, \ldots, sk_u)$ and $\overrightarrow{pk} = (pk_1, \ldots, pk_u)$ as the output.
- $s_i \leftarrow ASig.Sign(sk_i, m_i)$: As in Definition 1 signature generation algorithm.
- $\sigma_{1,u} \leftarrow ASig.Agg(\{pk_i, m_i, s_i\}_{i=1}^{u})$: The aggregation algorithm takes $\{pk_i, m_i, s_i\}_{i=1}^{u}$ as the input. It combines individual signatures $s_i, 1 \leq i \leq u$ and returns an aggregate signature $\sigma_{1,u}$ as the output.
- $c \leftarrow ASig.Ver(\{pk_i, m_i\}_{i=1}^{u}, \sigma_{1,u})$: The verification algorithm takes $\{pk_i,\ m_i\}_{i=1}^{u}$ and $\sigma_{1,u}$ as the input. It outputs a bit c, with $c = 1$ meaning *valid* and $c = 0$ meaning *invalid*.

The EU-CMA experiment for $ASig$ is a straightforward extension of Definition 2, in which $\mathcal{A}$ is required to produce a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

forgery under a public key $pk \in \overrightarrow{pk}$ that is not under his control during the experiment (see [8] for details).

We define the $PISB$ syntax as below.

Definition 4 $PISB$ signature scheme is a tuple of four algorithms $(Kg, Init, Sign, Ver)$ defined as follows:

- $(SK, PK) \leftarrow PISB.Kg(1^\kappa)$: Given the security parameter 1^κ, the key generation algorithm runs $(\overrightarrow{sk}, \overrightarrow{pk}) \leftarrow ASig.Kg(1^\kappa)$ and $(\overline{sk}, \overline{pk}) \leftarrow Sig.Kg(1^\kappa)$. It returns a private/public key pair $SK = (\overrightarrow{sk}, \overline{sk})$ and $PK = (\overrightarrow{pk}, \overline{pk})$, respectively, as the output.
- $\overrightarrow{V} \leftarrow PISB.Init(\overrightarrow{M}, \overrightarrow{sk}, PK)$: The initialization algorithm takes messages $\overrightarrow{M} = (m_1, \ldots, m_u)$, $\overrightarrow{sk}$ and PK as the input. It computes individual signatures $s_i \leftarrow ASig.Sign(sk_i, m_i), i = 1, \ldots, u$ and returns $\overrightarrow{V} = (\overrightarrow{M}, \overrightarrow{S} = \{s_i\}_i^u, PK)$ as the output.
- $\gamma \leftarrow PISB.Sig(\overline{sk}, \overrightarrow{m}, \overrightarrow{V})$: The signing algorithm takes $\overline{sk}$, messages $\overrightarrow{m} \in \overrightarrow{M}$ and $\overrightarrow{V}$ as the input. It aggregates corresponding signatures $s_i \in \overrightarrow{S}$ on $\overrightarrow{m} \in \overrightarrow{M}$ via $ASig.Agg$ under $\overrightarrow{pk}$, and then computes an individual signature s' on σ via $Sig.Sign$ under $\overline{pk}$. It returns the signature $\gamma \leftarrow (\sigma, s')$ as the output.
- $c \leftarrow PISB.Ver(PK, \overrightarrow{m}, \gamma)$: The verification algorithm takes PK, $\overrightarrow{m}$ and γ as the input. If σ on $\overrightarrow{m}$ and s' on σ under $\overrightarrow{pk}$ and $\overline{pk}$ are valid, respectively, then it outputs a bit $c = 1$ meaning *valid*; otherwise $c = 0$ meaning *invalid*.

Condensed-RSA (i.e., $C\text{-}RSA$) [3], [7] aggregates RSA signatures computed under *the same private key*. Hence, it is used for *single querier (signer)* applications.

Definition 5 $C\text{-}RSA$ is a tuple of three algorithms (Kg, Sig, Ver) defined as follows:

- $(sk, pk) \leftarrow C\text{-}RSA.Kg(1^\kappa)$: Given the security parameter 1^κ, the key generation algorithm generates a RSA private/public key pair. That is, it randomly generates two large primes (p, q) and computes $n = p \cdot q$. The public and secret exponents $(e, d) \in Z_n^*$ satisfies $e \cdot d \equiv 1 \bmod \phi(n)$, where $\phi(n) = (p-1)(q-1)$. The key generation algorithm returns $sk \leftarrow (n, d)$ and $pk \leftarrow (n, e)$ as the output.
- $\sigma \leftarrow C\text{-}RSA.Sig(sk, \overrightarrow{m})$: Given sk and messages $\overrightarrow{m} = (m_1, \ldots, m_l)$, the signing algorithm returns a signature $\sigma \leftarrow \prod_{j=1}^{l} s_j \bmod n$ as the output, where $s_j \leftarrow [H(m_j)]^d \bmod n$ for $j = 1, \ldots, l$. H is a full domain hash function (e.g., [52]) defined as $H : \{0,1\}^* \rightarrow Z_n$.
- $c \leftarrow C\text{-}RSA.Ver(pk, \overrightarrow{m}, \sigma)$: Given $pk = (n, e)$, $\overrightarrow{m}$ and σ, if $\sigma^e = \prod_{j=1}^{l} H(m_j) \bmod n$ then the signature verification algorithm returns bit $c = 1$ else $c = 0$.

A sequential aggregate signature (e.g., [16]) performs signature generation and verification operations in a specific order. One example is $SA\text{-}RSA$ [16], in which the signature generation and aggregation operations are performed together. We define $SA\text{-}RSA$ as below:

Definition 6 $SA\text{-}RSA$ [16] is a tuple of three algorithms $(Kg, ASign, Ver)$ defined as follows:

- $(\overrightarrow{sk}, \overrightarrow{pk}) \leftarrow SA\text{-}RSA.Kg(1^\kappa)$: Given the security parameter 1^κ and a set of signers $\mathbb{U} = \{1, \ldots, u\}$, the key generation algorithm generates a RSA private/public key pair $sk_i \leftarrow (n_i, d_i)$ and $pk_i \leftarrow (n_i, e_i)$, ensuring that $2^{k-1}(1 + (i-1)/u) \leq n_i < 2^{k-1}(1 + i/u)$, where $k = |n_i|$ for $i = 1, \ldots, u$. It returns a private/public key pair $\overrightarrow{sk} \leftarrow \{n_i, d_i\}_{i=1}^u$ and $\overrightarrow{pk} \leftarrow \{n_i, e_i\}_{i=1}^u$ as the output.
- $\sigma_{1,u} \leftarrow SA\text{-}RSA.ASig(sk_u, \{m_i\}_{i=1}^{u-1}, m_u, \{pk_i\}_{i=1}^{u-1}, pk_u, \sigma_{1,u-1})$: The signer u receives aggregate signature $\sigma_{1,u-1}$ on messages $\{m_i\}_{i=1}^{u-1}$ under public keys $\{pk_i\}_{i=1}^{u-1}$. The signer u first verifies $\sigma_{1,u-1}$ with the verification algorithm $SA\text{-}RSA.Ver$. If it succeeds, the signer u computes the signature as $h_u = H(\overrightarrow{m}||\overrightarrow{pk})$ and $y_u = h_u + \sigma_{1,u-1}$. The sequential aggregate signature algorithm outputs $\sigma_{1,u} \leftarrow y_u^{d_u} \bmod n_u$.
- $c \leftarrow SA\text{-}RSA.Ver(\overrightarrow{m}, \overrightarrow{pk}, \sigma_{1,u})$: Given $\sigma_{1,u}$ on $\overrightarrow{m}$ under public keys $\overrightarrow{pk} = \{n_i, e_i\}_{i=1}^u$, first check $0 \leq \sigma_{1,u} \leq n_u$. If $gcd(\sigma_{1,u}, n_u) = 1$ then $y_u \leftarrow \sigma_{1,u}^{e_u} \bmod n_u$ else $y_u \leftarrow \sigma_{1,u}$. Compute $h_u \leftarrow H(\overrightarrow{m}||\overrightarrow{pk})$ and $\sigma_{1,u-1} \leftarrow (y_u - h_u) \bmod n_u$. Verify signatures recursively as described in this verification algorithm until the base case $u = 1$, in which check $(\sigma_{1,1} - h_1) \bmod n_1 = 0$ where $h_1 \leftarrow (m_1||pk_1)$. If it holds return $c = 1$ else $c = 0$.

In our $PISB\text{-}CSA\text{-}RSA$ scheme, we use a (simplified) *single signer (and aggregator) instantiation* of $SA\text{-}RSA$ [16].

3 Security Model

We first give the security model of $PISB$ schemes. We then give an analysis on the aggregate signature extraction [8], [10] and the immutable aggregate signature constructions.

3.1 PISB Security Model

Our security model reflects how $PISB$ system model works. That is, our security model formally captures the immutability of aggregate signatures for the $EU\text{-}CMA$ experiment, which we call Immutable-$EU\text{-}CMA$ ($I\text{-}EU\text{-}CMA$) experiment.

Definition 7 $I\text{-}EU\text{-}CMA$ for $PISB$ is defined as follows:

- *Setup.* Challenger algorithm $\mathcal{B}$ runs $(SK, PK) \leftarrow PISB.Kg(1^\kappa)$ and provides PK to the adversary $\mathcal{A}$.
- *Queries.* $\mathcal{A}$ queries $\mathcal{B}$ on any message $\overrightarrow{m}_j = (m_{j,1}, \ldots, m_{j,u})$ of her choice for $j = 1, \ldots, q_s$. $\mathcal{B}$ replies each query j with a signature γ_j computed under PK.
- *Forgery.* $\mathcal{A}$ outputs a forgery (m^*, γ^*) and wins the $EU\text{-}CMA$ experiment, if
  - (i) $PISB.Ver(PK, m^*, \gamma^*) = 1$,
  - (ii) $m^* \not\subseteq \{\overrightarrow{m}_j\}_{j=1}^{q_s}$ or $\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq ||_{k \in I} \overrightarrow{m}_k$

  That is, (i) the forgery is valid, (ii) m^* has not been queried previously, or it is a subset and/or any combination of previously queried data items $(\overrightarrow{m}_1, \ldots, \overrightarrow{m}_{q_s})$.

$PISB$ is (t, q_s, ϵ)-$I\text{-}EU\text{-}CMA$ secure, if no $\mathcal{A}$ in time t making at most q_s signature queries has an advantage at least with probability ϵ in the above experiment.

3.2 Aggregate Signature Extraction and Signature Immutability

u-aggregate signature extraction problem (afterwards referred as AE problem for the brevity) was first introduced by Boneh et. al. in [8] to ensure the security of pairing-based BGLS aggregate signature schemes. However, this problem can be generalized for other types of aggregate signatures.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

Intuitively, the difficulty of aggregate signature extraction implies that for a given aggregate signature $\sigma_{1,u}$ computed from u individual signatures, it is difficult to extract these individual signatures $\sigma_1, \ldots, \sigma_u$ provided that only $\sigma_{1,u}$ is known to the extractor. Moreover, it should be difficult to extract any aggregate signature subset σ' from a given of $\sigma_{1,u}$.

Definition 8 AE experiment for $ASig$ is defined as follows:

- *Setup.* Challenger algorithm $\mathcal{B}$ runs $(\overrightarrow{sk}, \overrightarrow{pk}) \leftarrow ASig.Kg(1^\kappa)$ and provides $\overrightarrow{pk}$ to algorithm $\mathcal{A}$.
- *Queries.* $\mathcal{A}$ queries $\mathcal{B}$ on any *batch message* $\overrightarrow{m}_j = (m_{j,1}, \ldots, m_{j,u})$ of her choice for $j = 1, \ldots, q_s$. $\mathcal{B}$ replies each query j with a signature σ_j computed under $\overrightarrow{pk}$.
- *Aggregate Extraction.* $\mathcal{A}$ outputs a message-signature pair (m^*, σ'), where $m^* = (m_1^*, \ldots, m_k^*), 1 \leq k \leq u$ and wins the AE experiment, if
  - (i) $ASig.Ver(\{pk_i, m_i^*\}_{i\in\{1,\ldots,k\}}, \sigma') = 1$,
  - (ii) $\exists I' \subseteq \{1, \ldots, q_s\} : m^* \subseteq ||_{k\in I'} \overrightarrow{m}_k$,
  - (iii) $\forall I \subseteq \{1, \ldots, q_s\} : [m^* || (||_{j\in I} \overrightarrow{m}_j)] \neq \{\overrightarrow{m}_l\}_{l=1}^{q_s}$

  That is, (i) the forgery is valid, (ii) m^* is a subset of previously queried or some combination of previously queried batch messages, and (iii) if m^* is combined with any previously queried or a combination of previously queried batch messages, the combination is not equal to one of the previously queried batch message itself. That is, the extracted signature is not simply derived from previously queried signatures (which implies signature mutability), and therefore the aggregate signature extraction is non-trivial.

$ASig$ is (t, q_s, ϵ)-AE secure, if no $\mathcal{A}$ in time t making at most q_s signature queries has an advantage at least with probability ϵ in the above experiment.

Initially, Boneh's AE problem is given as an intractability assumption without a proof. Later, Coron et. al. in [10] proved that Boneh's AE problem for BGLS scheme is equivalent to the Computational Diffie Hellman Assumption (CDH) [53]. Yavuz et. al. in [21] analyzed log truncation problem for forward-secure and aggregate signatures [20], [54], and produced formal proofs with AE argument for only DLP-based schemes [54]. A related problem for one-way accumulators for RSA have been considered in [39], which extends to other aggregate RSA variants (e.g., C-RSA [7] and SA-RSA [16]).

4 The Proposed Schemes

In this section, we describe our proposed schemes. For each $PISB$ scheme, we first give the intuition behind the scheme followed by its detailed description.

4.1 PISB-Generic Scheme

Our generic scheme relies on *a very simple observation*: It is possible to guarantee the immutability of an aggregate signature by simply computing a standard digital signature on it. The server can sign the aggregate signature with his private key and define the immutable signature as a signature pair.

$PISB$-$Generic$ slightly increases the signature size, since a secondary signature is transmitted along with the aggregate signature. However, this is actually *much more communication efficient* than GQ-based and SKROOT-based methods in [3], [7]. That is, a secondary standard signature (e.g., ECDSA [55] with 40 bytes) is much smaller than cryptographic values transmitted (e.g., up to 9 KB) to achieve the immutability in [3], [7]. $PISB$-$Generic$ also allows the server to choose *any* signature scheme to provide the immutability. For instance, the server may use ECDSA tokens [24] or offline/online signatures [26], which enable faster response times in demand peaks via pre-computability (as discussed in Section 1.3). This flexibility makes $PISB$-$Generic$ more efficient at the server side than previous alternatives (see Table 1).

The $PISB$-$Generic$ algorithms, as a realization of the $PISB$ syntax (Definition 4) in the ODB settings, are as below.

1) $(SK, PK) \leftarrow PISB\text{-}Generic.Kg(1^\kappa)$: Execute $(\overrightarrow{sk}, \overrightarrow{pk}) \leftarrow ASig.Kg(1^\kappa)$ for data owners $\mathbb{U} = \{1, \ldots, u\}$. Execute $(\overline{sk}, \overline{pk}) \leftarrow Sig.Kg(1^\kappa)$ for the server. The system private and public keys are $SK = (\overrightarrow{sk}, \overline{sk})$ and $PK = (\overrightarrow{pk}, \overline{pk})$, respectively.

2) $\overrightarrow{V} \leftarrow PISB\text{-}Generic.Init(\overrightarrow{M}, \overrightarrow{sk}, PK)$: Let messages $\overrightarrow{M} = \{\overrightarrow{m}_1, \ldots, \overrightarrow{m}_u\}$ be database elements to be outsourced, where each $\overrightarrow{m}_i = (m_{i,1}, \ldots, m_{i,l})$ belongs to the data owner $1 \leq i \leq u$. Each data owner i computes $s_{i,j} \leftarrow ASig.Sign(sk_i, m_{i,j})$ for $i = 1, \ldots, u$ and $j = 1, \ldots, l$. Set $\overrightarrow{V} \leftarrow (\overrightarrow{M}, \overrightarrow{S}, PK)$ and provide $\overrightarrow{V}$ to the server, where $\overrightarrow{S} = \{s_{i,j}\}_{i=1,j=1}^{u,l}$.

3) $\gamma \leftarrow PISB\text{-}Generic.Sign(\overline{sk}, \overrightarrow{m}, \overrightarrow{V})$: The server receives a multiple-signer query $\overrightarrow{m} = \{m_1, \ldots, m_k\}$ on a subset of k data owners $U \subseteq \mathbb{U}$. Fetch the corresponding public key and signatures on $\overrightarrow{m}$ from $\overrightarrow{V}$ as $V \leftarrow \{pk_i, m_{i,j}, s_{i,j}\}_{\forall i\in U, \exists j: m_{i,j}\in\overrightarrow{M}}$ and compute $\sigma \leftarrow ASig.Agg(V)$. Also compute $s' \leftarrow Sig.Sign(\overline{sk}, \sigma)$ and set $\gamma \leftarrow (\sigma, s')$.

4) $c \leftarrow PISB\text{-}Generic.Ver(PK, \overrightarrow{m}, \gamma)$: Given $\gamma = (\sigma, s')$ and $pk \leftarrow \{pk_i\}_{\forall i\in U}$, if $Sig.Ver(\overline{pk}, s', \gamma) = 1$ and $ASig.Ver(pk, \overrightarrow{m}, \sigma) = 1$ hold return $c = 1$, else $c = 0$.

Remark 1 One may further strengthen $PISB$ constructions by involving index numbers and timestamps, which are needed if the application is sensitive to the order of data items and freshness of the data query.

(i) For instance, in $PISB$-$Generic$, the server may compute the protection signature as $s' \leftarrow Sig.Sign(\overline{sk}, \sigma||ts_\sigma)$, where ts_σ is the timestamp of the protection signature. This ensures that each protection signature is unique and achieves the freshness. (ii) During the initialization phase, each data owner and data items of each owner are associated with indexes indicating their order in the aggregate signature. That is, each data owner i computes $s_{i,j} \leftarrow ASig.Sign(sk_i, m_{i,j}||i||j)$ for $i = 1, \ldots, u$ and $j = 1, \ldots, l$. This ensures the order of data items if the query response is sensitive to such an order.

4.2 PISB-CSA-RSA Scheme

An effective way to provide the signature immutability is to compute the protection signature by replacing the standard signature in $PISB$-$Generic$ with an aggregate signature (no extra protection signature is transmitted). However, this method is *not* applicable to aggregate signatures such as C-RSA, in which only the signatures computed with the same private key, can be aggregated (also called as *single signer* aggregate signature). Recall that C-RSA cannot aggregate signatures belonging to different signers, since an RSA

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

modulus n can not be safely shared among multiple signers (this leads to the factorization of n, exposing the private keys [56]). Hence, despite $C\text{-}RSA$ is an efficient scheme, its immutable variants (e.g., [3], [7]) are inefficient as discussed in Section 1.2.

It is highly desirable to construct a scheme that can compute an aggregate RSA signature involving both a data owner and the server (without exposing their private keys via the factorization of modulo). Our *main observation* is that, this goal can be achieved by levering the sequential aggregate signatures from trapdoor permutations (e.g., $SA\text{-}RSA$ [16], as defined in Definition 6) together with $C\text{-}RSA$. We call our new scheme that exploits this observation as $PISB\text{-}CSA\text{-}RSA$.

In $PISB\text{-}CSA\text{-}RSA$, the data owner computes RSA signatures $s_1, \ldots, s_l$ on $m_1, \ldots, m_l$ with her keys (n, d). During the query phase, the server computes a $C\text{-}RSA$ signature σ' by aggregating RSA signatures. The server then uses $SA\text{-}RSA$ to compute an immutable aggregate signature γ on $m_1, \ldots, m_l$ with his keys $(\overline{n}, \overline{d})$ by aggregating it on σ'. The public key of the system is $(\langle n, e\rangle, \langle \overline{n}, \overline{e}\rangle)$. The verification order of the client is with $SA\text{-}RSA$ under $(\overline{n}, \overline{e})$ for γ and then with $C\text{-}RSA$ under (n, e) for σ'.

$PISB\text{-}CSA\text{-}RSA$ is an instantiation of $PISB\text{-}Generic$, in which the multiple signer $ASig$ is replaced with the single signer $C\text{-}RSA$, and the protection signature Sig is replaced with a simplified variant of $SA\text{-}RSA$. Since $SA\text{-}RSA$ can aggregate on the $C\text{-}RSA$ output, $PISB\text{-}CSA\text{-}RSA$ immutable signature does not include an extra signature component.

The $PISB\text{-}CSA\text{-}RSA$ algorithms are defined below. Note that the algorithm is executed only for a single signer case.

1) $(SK, PK) \leftarrow PISB\text{-}CSA\text{-}RSA.Kg(1^\kappa)$: The data owner executes $(sk, pk) \leftarrow C\text{-}RSA.Kg(1^k)$, where $sk = (n, d)$ and $pk = (n, e)$. The server generates a RSA private/public key pair $\overline{sk} \leftarrow (\overline{n}, \overline{d})$ and $\overline{pk} \leftarrow (\overline{n}, \overline{e})$, $n < \overline{n}$. The system private/public key are $SK \leftarrow (sk, \overline{sk})$ and $PK \leftarrow (pk, \overline{pk})$.

2) $\overrightarrow{V} \leftarrow PISB\text{-}CSA\text{-}RSA.Init(\overrightarrow{m}, sk)$: The data owner computes an individual signature $s_j \leftarrow [H(m_j)]^d \bmod n$ for $j = 1, \ldots, l$, where $\overrightarrow{m} = (m_1, \ldots, m_l)$. The data owner sets the message-signature pairs as $\overrightarrow{V} \leftarrow (\overrightarrow{m}, \overrightarrow{S})$ and provide $\overrightarrow{V}$ to the server, where $\overrightarrow{S} = (s_1, \ldots, s_l)$.

3) $\gamma \leftarrow PISB\text{-}CSA\text{-}RSA.Sign(\overline{sk}, \overrightarrow{m}, \overrightarrow{V})$: The server receives a single-signer query $\overrightarrow{m} = (m_1, \ldots, m_l)$. It fetches the corresponding signatures $(s_1, \ldots, s_l)$ on $\overrightarrow{m}$ from $\overrightarrow{V}$ and computes $\sigma' \leftarrow \prod_{j=1}^{l} s_j \bmod n$. It then computes $h \leftarrow H(\overrightarrow{m}||\overline{pk})$, $y \leftarrow (h + \sigma') \bmod \overline{n}$ and $\gamma \leftarrow y^{\overline{d}} \bmod \overline{n}$.

4) $c \leftarrow PISB\text{-}CSA\text{-}RSA.Ver(PK, \overrightarrow{m}, \gamma)$: Given γ, the verifier computes $y' \leftarrow \gamma^{\overline{e}} \bmod \overline{n}$ and $\sigma' \leftarrow (y' - h') \bmod \overline{n}$, where $h' \leftarrow H(\overrightarrow{m}||\overline{pk})$. If $C\text{-}RSA.Ver(pk, \overrightarrow{m}, \sigma') = 1$ then return $c = 1$ else $c = 0$.

Remark 2 In $PISB\text{-}CSA\text{-}RSA$, we use a *simplified* $SA\text{-}RSA$ variant [16] with the following properties: *(i)* $SA\text{-}RSA$ is used in a *single signer* setting (the server as the signer and aggregator). *(ii)* The public key correctness controls (e.g., range check and gcd control) are not required, since the public keys are *already certified* in our system model. That is, n_i belongs to a legitimate signer and $gcd(e_i, \phi(n_i)) = 1$ holds. This retains the computational efficiency of traditional small RSA exponents.

4.3 PISB-RP Scheme

To achieve a minimum end-to-end delay, it is important to minimize the server's online signature generation overhead (i.e., the computation overhead of protection signature). One may consider directly adapting pre-computation methods (e.g., [24] or offline/online signatures (e.g., [23]) to $PISB$ setting. However, as discussed in Section 1.3, special pre-computation methods are not applicable to various $PISB$ instantiations such as $C\text{-}RSA$ and pairing-based constructions; and offline/online signatures introduce extremely large public key and signatures sizes. Hence, none of these methods are suitable to accelerate signature generation in $PISB$ schemes.

To address these limitations, we developed Random Precomputed PISB ($PISB\text{-}RP$) scheme. $PISB\text{-}RP$ can transform any AE secure aggregate signature scheme (see Section 3.2) into an immutable pre-computable aggregate signature. The intuition behind our scheme is to inject a random component (along with its signature) into each aggregate signature, which can be computed independent from messages to be signed. Randomness prevents an adversary to create mutations from existing signatures, while the message independency enables signature pre-computability to gain performance advantages. We achieve this by generating a random number r and its signature s (i.e., the protection signature), which can be aggregated into any future aggregate signature σ by the server. The pair (r, s) can be generated and stored during the offline phase by the server, or the data owner may generate and transmit them to the server (only for single-user cases). During the online phase, the server aggregates protection signature s into σ, which is more efficient than computing s itself.

Notice that $PISB\text{-}RP$ does not rely on costly one-time signatures (e.g., 3-5 KB signature and public key overhead for per message), and therefore is much more communication and storage efficient than $PISB\text{-}Generic$ instantiated with offline/online signatures (e.g., [27]). It also does not compute online signatures and does not transmit an extra protection signature. Hence, it is more communication and computation efficient than $PISB\text{-}Generic$. However, unlike $PISB\text{-}Generic$ (but like offline/online and other non-generic pre-computed tokens [24]), it requires storing a pair as (r, s) for each message to be signed at the server side.

The $PISB\text{-}RP$ algorithms are defined below.

1) $(SK, PK) \leftarrow PISB\text{-}RP.Kg(1^\kappa)$: Execute $(\overrightarrow{sk}, \overrightarrow{pk}) \leftarrow ASig.Kg(1^\kappa)$ for $\mathbb{U} = \{1, \ldots, u\}$. Execute $(\overline{sk}, \overline{pk}) \leftarrow ASig.Kg(1^\kappa)$ for the server. The system private and public keys are $SK = (\overrightarrow{sk}, \overline{sk})$ and $PK = (\overrightarrow{pk}, \overline{pk})$, respectively.

2) $\overrightarrow{V} \leftarrow PISB\text{-}RP.Init(\overrightarrow{M}, \overrightarrow{sk}, PK)$: Identical to that of $PISB\text{-}Generic.Init$.

3) $\gamma \leftarrow PISB\text{-}RP.Sign(\overline{sk}, \overrightarrow{m}, \overrightarrow{V})$: It has two phases:

- *Offline Phase*: The server pre-computes (r, s) to be used in online phase, where $r \xleftarrow{\$} \{0,1\}^\kappa$, $s \leftarrow ASig.Sign(\overline{sk}, r)$ and store (r, s).
- *Online Phase*: The server receives a multiple-signer query $\overrightarrow{m} = \{m_1, \ldots, m_k\}$ on a subset of k data owners $U \subseteq \mathbb{U}$. Fetch the corresponding public key and signatures on $\overrightarrow{m}$ from $\overrightarrow{V}$ as $V \leftarrow \{pk_i, m_{i,j}, s_{i,j}\}_{\forall i \in U, \exists j : m_{i,j} \in \overrightarrow{M}}$ and compute $\sigma' \leftarrow ASig.Agg(V)$. Also fetch a pre-computed pair (r, s) and set $\gamma \leftarrow (\sigma, r)$, where $\sigma \leftarrow ASig.Agg(\overrightarrow{pk}, \langle \overrightarrow{m}, r\rangle, \langle \sigma', s\rangle)$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

4) $c \leftarrow PISB\text{-}RP.Ver(PK, \overrightarrow{m}, \gamma)$: Given $\gamma = (\sigma, r)$ and $\overrightarrow{pk} \leftarrow (\{pk_i\}_{\forall i \in U}, \overline{pk}) \subseteq PK$, if $|r| = \kappa$ and $ASig.Ver(pk, \langle \overrightarrow{m}, r \rangle, \sigma) = 1$ hold return $c = 1$, else $c = 0$.

We now discuss two specific instantiations of $PISB\text{-}RP$. For the sake of brevity, we do not give the full descriptions, but only explain how signing and aggregation are performed (other steps are straightforward to derive from $PISB\text{-}RP$).

• *BGLS-based*: At the offline phase (following the notation given in [8]), the server generates $r \xleftarrow{\$} \{0,1\}^\kappa$, $h \leftarrow \overline{h}(r), s \leftarrow h^x$ and store (r, s). At the online phase, given σ' on $\overrightarrow{m}$, the server computes $\sigma \leftarrow \sigma' \cdot s$ and set $\gamma \leftarrow (\sigma, r)$.

Notice that, compared to iBGLS [7], [12] and $PISB\text{-}Generic$, the server saves one full exponentiation (or one full scalar multiplication [57], [58] if it is implemented with ECC) during the online phase with this instantiation of $PISB\text{-}RP$. The only online computational cost for the server is a modular addition operation and hash operation, whose costs are negligible. However, the server stores $|q| + \kappa$ bits for per message to be signed during the offline phase.

• *C-RSA based*: This instantiation only works for the *single signer* case. Unlike BGLS-based instantiation, not the server but the data owner generates pre-computed pairs and then provides them to the server to be used during the online phase (the server cannot aggregate his RSA signature on r over $C\text{-}RSA$ signature of the data owner): At the initialization phase $PISB\text{-}RP.Init$, the data owner generates $r \xleftarrow{\$} \{0,1\}^\kappa$, $s \leftarrow [H(r)]^d \bmod n$ and gives (r, s) to the server (this corresponds to the *offline phase* of $PISB\text{-}RP.Sign$ executed by the server). At the signature generation (online) phase of $PISB\text{-}RP.Sign$, given aggregate signature σ' on $\overrightarrow{m}$, the server computes $\sigma \leftarrow \sigma' \cdot s \bmod n$ and set $\gamma \leftarrow (\sigma, r)$.

This instantiation provides the lowest end-to-end computation delay among all the compared schemes, since the online computational overhead of server and client are only a few modular multiplication plus hash operations.

5 Security Analysis

We prove that $PISB$ schemes are $I\text{-}EU\text{-}CMA$ secure in Theorem 1 and Theorem 2 (in ROM [50]). We ignore terms that are negligible in terms of κ.

Theorem 1 *$PISB\text{-}Generic$ is (t, q_s, ϵ)-$I\text{-}EU\text{-}CMA$ secure, if $ASig$ is (t', q_s, ϵ)-$EU\text{-}CMA$ secure and Sig is (t', q_s, ϵ)-$EU\text{-}CMA$ secure, where $t' = O(t) + q_s \cdot (Op + Op')$ and (Op, Op') denote the cost of signature generation for $ASig$ and Sig, respectively.*

Proof: Suppose algorithm $\mathcal{A}$ breaks (t, q_s, ϵ)-$I\text{-}EU\text{-}CMA$ secure $PISB\text{-}Generic$. We then construct a simulator $\mathcal{B}$, which breaks (t', q_s, ϵ)-$EU\text{-}CMA$ secure $ASig$ or (t', q_s, ϵ)-$EU\text{-}CMA$ secure Sig by using $\mathcal{A}$ as subroutine.

We set the $EU\text{-}CMA$ experiments for $ASig$ and Sig. $\mathcal{B}$ is given a $ASig$ public key $\overrightarrow{pk}$ and a Sig public key $\overline{pk}$ as the input, where $(\overrightarrow{sk}, \overrightarrow{pk}) \leftarrow ASig.Kg(1^\kappa)$ and $(\overline{sk}, \overline{pk}) \leftarrow Sig.Kg(1^\kappa)$. $\mathcal{B}$ is given an access to $ASig.Sign$ and $Sig.Sign$ oracles under $\overrightarrow{sk}$ and $\overline{sk}$ up to q_s signature queries on both, respectively (as in Definition 2).

We then set the $I\text{-}EU\text{-}CMA$ experiment for $PISB\text{-}Generic$, in which $\mathcal{B}$ executes $\mathcal{A}$ as follows:

- *Setup:* Given $(\overrightarrow{pk}, \overline{pk})$, $\mathcal{B}$ sets the $PISB\text{-}Generic$ public key $PK \leftarrow (\overrightarrow{pk}, \overline{pk})$ as in $PISB\text{-}Generic.Kg$ algorithm. By Definition 7, $\mathcal{B}$ gives PK to $\mathcal{A}$ and also permits $\mathcal{A}$ to make q_s $PISB\text{-}Generic$ signature queries.
- Queries: $\mathcal{A}$ queries $\mathcal{B}$ on messages $\overrightarrow{m}_j = (m_{j,1}, \ldots, m_{j,u})$ of her choice for $j = 1, \ldots, q_s$. $\mathcal{B}$ handles queries as follows:
  a) Given $\mathcal{A}$'s j-th query $\overrightarrow{m}_j$, $\mathcal{B}$ queries $ASig.Sign$ oracle on $\overrightarrow{m}_j$ under $\overrightarrow{pk}$. The $ASig.Sign$ oracle returns $s_{j,i} \leftarrow ASig.Sign(sk_i, m_{j,i})$ for $i = 1, \ldots, u$. $\mathcal{B}$ then computes the aggregate signature as $\sigma_j \leftarrow ASig.Agg(\overrightarrow{pk}, \overrightarrow{m}_j, s_{j,1}, \ldots, s_{j,u})$. This step is identical to $PISB\text{-}Generic.Init$ algorithm, where $\overrightarrow{M}$ in this experiment is comprised of u vectors each with q_s data items.
  b) $\mathcal{B}$ queries $Sig.Sign$ oracle on σ_j under $\overline{pk}$. The $Sig.Sign$ oracle returns $s'_j \leftarrow Sig.Sign(\overline{sk}, \sigma_j)$ (as in $PISB\text{-}Generic.Sign$ algorithm, executed by the server). $\mathcal{B}$ replies $\mathcal{A}$ with $\gamma_j = (\sigma_j, s'_j)$.
- Forgery of $\mathcal{A}$: $\mathcal{A}$ outputs a forgery $(m^*, \beta^* = \langle \sigma^*, s'^* \rangle)$ and wins the $I\text{-}EU\text{-}CMA$ experiment if
  (i) $PISB\text{-}Generic.Ver(PK, m^*, \beta^*) = 1$,
  (ii) $m^* \not\subseteq \{\overrightarrow{m}_1, \ldots, \overrightarrow{m}_{q_s}\}$ or $\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq ||_{k \in I} \overrightarrow{m}_k$

If $\mathcal{A}$ loses in the $I\text{-}EU\text{-}CMA$ experiment then $\mathcal{B}$ loses in the $EU\text{-}CMA$ experiments for $ASig$ and Sig, and $\mathcal{B}$ *aborts*. Otherwise, $\mathcal{B}$ proceeds for two possible forgeries as follows:

a) If $m^* \not\subseteq \{\overrightarrow{m}_1, \ldots, \overrightarrow{m}_{q_s}\}$ then $\mathcal{B}$ returns the forgery (m^*, σ^*) against $ASig$, which is non-trivial since $\mathcal{B}$ did not ask m^* to $ASig.Sign$. This forgery is valid since $PISB\text{-}Generic.Ver(PK, m^*, \beta^*) = 1$ implies $ASig.Ver(\overrightarrow{pk}, m^*, \sigma^*) = 1$.
b) If $\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq ||_{k \in I} \overrightarrow{m}_k$ then $\mathcal{B}$ returns the forgery (σ^*, s'^*) against Sig, which is non-trivial since $\mathcal{B}$ did not ask σ^* to $Sig.Sign$. This forgery is valid since $PISB\text{-}Generic.Ver(PK, m^*, \beta^*) = 1$ implies $Sig.Ver(\overline{pk}, \sigma^*, s'^*) = 1$.

The execution time of $\mathcal{B}$ is that of $\mathcal{A}$ plus the time required to handle $\mathcal{A}$'s queries. That is, for each query of $\mathcal{A}$, $\mathcal{B}$ requests one $ASig$ and Sig signature, whose total costs for handling q_s queries is $q_s \cdot (Op + Op')$. Hence, $t' = O(t) + q_s \cdot (Op + Op')$.

$\mathcal{A}$ does not abort the during the query phase, as the simulation of $\mathcal{B}$ is perfectly indistinguishable. That is, the real and simulated views of $\mathcal{A}$ are identical, and each value in these views are computed identically as described during the experiment. The probability that $\mathcal{A}$ wins the experiment without querying $\mathcal{B}$ is negligible in terms of κ. Therefore, $\mathcal{B}$ wins with the probability ϵ that $\mathcal{A}$ wins.

Theorem 2 *$PISB\text{-}CSA\text{-}RSA$ is (t, q_s, ϵ)-$I\text{-}EU\text{-}CMA$ secure if RSA signature scheme is $(t', (2 \cdot l)q_s, \epsilon)$-$EU\text{-}CMA$ secure, where $t' = O(t) + 2(l \cdot q_s)Exp$ and Exp and l denote modular exponentiation and number of messages in a single $PISB\text{-}CSA\text{-}RSA$ query, respectively.*

Proof: Suppose algorithm $\mathcal{A}$ breaks (t, q_s, ϵ)-$I\text{-}EU\text{-}CMA$ secure $PISB\text{-}CSA\text{-}RSA$. We construct a simulator $\mathcal{B}$ breaking $(t', (2 \cdot l)q_s, \epsilon)$-$EU\text{-}CMA$ RSA by using $\mathcal{A}$ as subroutine.

We set two separate $EU\text{-}CMA$ experiments for $\mathcal{B}$, in which it is given RSA public keys $pk = (n, e)$ and $\overline{pk} = (\overline{n}, \overline{e})$ and provided signature oracles under their corresponding private

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

keys $sk = (n, d)$ (i.e., oracle $\mathcal{O}_1$) and $\overline{sk} = (\overline{n}, \overline{d})$ (i.e., oracle $\mathcal{O}_2$), respectively. $\mathcal{B}$ will simulate $\mathcal{A}$'s signature queries via $(\mathcal{O}_1, \mathcal{O}_2)$. $\mathcal{B}$ then executes $\mathcal{A}$ for the $I\text{-}EU\text{-}CMA$ experiment for $PISB\text{-}CSA\text{-}RSA$ as follows:

- *Setup:* Given $(pk, \overline{pk})$, $\mathcal{B}$ sets the $PISB\text{-}CSA\text{-}RSA$ public key $PK \leftarrow (pk, \overline{pk})$ as in $PISB\text{-}CSA\text{-}RSA.Kg$ algorithm. By Definition 7, $\mathcal{B}$ gives PK to $\mathcal{A}$ and allows $\mathcal{A}$ to ask q_s $PISB\text{-}CSA\text{-}RSA$ signatures under PK.
- Queries: $\mathcal{A}$ queries $\mathcal{B}$ on messages $\overrightarrow{m}_j = (m_{j,1}, \ldots, m_{j,l})$ of her choice for $j = 1, \ldots, q_s$. $\mathcal{B}$ handles queries as follows:
  a) Given $\mathcal{A}$'s j-th query $\overrightarrow{m}_j$, $\mathcal{B}$ queries $\mathcal{O}_1$ on each $m_{j,i}$ and obtains corresponding s_i under pk for $i = 1, \ldots, l$ (as in $PISB\text{-}CSA\text{-}RSA.Init$, data owner).
  b) $\mathcal{B}$ computes $\sigma' \leftarrow \prod_{i=1}^{l} s_i \bmod n$, $h \leftarrow H(\overrightarrow{m}_j || \overline{pk})$ and $y \leftarrow h + \sigma'$. $\mathcal{B}$ queries $\mathcal{O}_2$ on y under $\overline{pk}$ and obtains γ (as in $PISB\text{-}CSA\text{-}RSA.Sign$, executed by the server).
- Forgery of $\mathcal{A}$: $\mathcal{A}$ outputs a forgery (m^*, γ^*) and wins the $I\text{-}EU\text{-}CMA$ experiment if
  (i) $PISB\text{-}CSA\text{-}RSA.Ver(PK, m^*, \gamma^*) = 1$,
  (ii) $m^* \not\subseteq \{\overrightarrow{m}_1, \ldots, \overrightarrow{m}_{q_s}\}$ or $\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq ||_{k \in I} \overrightarrow{m}_k$

  If $\mathcal{A}$ loses in the $I\text{-}EU\text{-}CMA$ experiment then $\mathcal{B}$ loses in the $EU\text{-}CMA$ experiments for RSA against $\mathcal{O}_1$ and $\mathcal{O}_2$, and $\mathcal{B}$ *aborts*. Otherwise, $\mathcal{B}$ computes $y^* \leftarrow (\gamma^*)^{\overline{e}} \bmod \overline{n}$ and $\sigma^* \leftarrow y^* - H(m^* || \overline{pk})$ and continues as follows:

a) If $m^* \not\subseteq \{\overrightarrow{m}_1, \ldots, \overrightarrow{m}_{q_s}\}$ then $\mathcal{B}$ returns the forgery (m^*, s^*) against $\mathcal{O}_1$, where s^* is computed from σ^* by removing the corresponding individual signatures of data items in m^* that have been queried before (if m^* is not a vector then use s^* itself). This forgery is non-trivial since $\mathcal{B}$ did not ask m^* to $\mathcal{O}_1$ during the experiment. $\mathcal{B}$ also returns the forgery (y^*, γ^*) against $\mathcal{O}_2$, which is non-trivial since $\mathcal{B}$ did not ask y^* to $\mathcal{O}_2$ during the experiment. Both forgeries are valid since $PISB\text{-}CSA\text{-}RSA.Ver(PK, m^*, \gamma^*) = 1$ implies m^* and y^* are valid under pk and $\overline{pk}$, respectively.
b) If $\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq ||_{k \in I} \overrightarrow{m}_k$ holds then $\mathcal{B}$returns the forgery (σ^*, γ^*) against $\mathcal{O}_2$. This forgery is valid and non-trivial as discussed the above.

The execution time and probability analysis are similar to Theorem 1 (i.e., the simulation is perfectly indistinguishable).

Theorem 3 *$PISB\text{-}RP$ is (t, q_s, ϵ)-$I\text{-}EU\text{-}CMA$ secure, if $ASig$ is (t', q_s, ϵ)-$EU\text{-}CMA$ secure and (t, q_s, ϵ)-AE secure, where $t' = O(t) + q_s \cdot Op$ and Op denotes the cost of signature generation for $ASig$.*

Proof: Suppose algorithm $\mathcal{A}$ breaks (t, q_s, ϵ)-$I\text{-}EU\text{-}CMA$ secure $PISB\text{-}RP$. We then construct a simulator $\mathcal{B}$, which breaks (t', q_s, ϵ)-$EU\text{-}CMA$ secure and (t, q_s, ϵ)-AE secure $ASig$ by using $\mathcal{A}$ as a subroutine.

We set the $EU\text{-}CMA$ experiments for $\mathcal{B}$, in which it is given $ASig$ public keys $(\overrightarrow{pk}, \overline{pk})$, where $(\overrightarrow{sk}, \overrightarrow{pk}) \leftarrow ASig.Kg(1^\kappa)$ and $(\overline{sk}, \overline{pk}) \leftarrow ASig.Kg(1^\kappa)$. $\mathcal{B}$ is also provided signature oracle $ASig.Sign$ under $PK = (\overrightarrow{pk}, \overline{pk})$. $\mathcal{B}$ will simulate $\mathcal{A}$'s signature queries via $ASig.Sign$. $\mathcal{B}$ also maintains two lists $\mathcal{LR}$ and $\mathcal{LD}$ that are used to keep track the queries during the experiment. $\mathcal{B}$ then executes $\mathcal{A}$ for the $I\text{-}EU\text{-}CMA$ experiment for $PISB\text{-}RP$ as follows:

- *Setup:* Given $(\overrightarrow{pk}, \overline{pk})$, $\mathcal{B}$ sets the $PISB\text{-}RP$ public key $PK \leftarrow (\overrightarrow{pk}, \overline{pk})$ as in $PISB\text{-}RP.Kg$ algorithm. By Definition 7, $\mathcal{B}$ gives PK to $\mathcal{A}$ and allows $\mathcal{A}$ to ask q_s $PISB\text{-}RP$ signatures under PK.
- Queries: $\mathcal{A}$ queries $\mathcal{B}$ on messages $\overrightarrow{m}_j = (m_{j,1}, \ldots, m_{j,u})$ of her choice for $j = 1, \ldots, q_s$. $\mathcal{B}$ handles queries as follows:
  a) $\mathcal{B}$ generates $r_j \leftarrow \{0,1\}^\kappa$ and sets $\widetilde{m}_j \leftarrow (\overrightarrow{m}_j, r_j)$. $\mathcal{B}$ inserts r_j and $\widetilde{m}_j$ into $\mathcal{LR}$ and $\mathcal{LD}$, respectively.
  b) $\mathcal{B}$ queries $ASig.Sign$ on $\widetilde{m}_j$ and obtains its corresponding signature σ_j. $\mathcal{B}$ sets $\gamma_j \leftarrow (\sigma_j, r_j)$ and returns γ_j to $\mathcal{A}$.
- Forgery of $\mathcal{A}$: $\mathcal{A}$ outputs a forgery (m^*, γ^*) and wins the $I\text{-}EU\text{-}CMA$ experiment if
  (i) $PISB\text{-}RP.Ver(PK, m^*, \gamma^*) = 1$,
  (ii) $m^* \not\subseteq \{\overrightarrow{m}_1, \ldots, \overrightarrow{m}_{q_s}\}$ or $\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq ||_{k \in I} \overrightarrow{m}_k$

  If $\mathcal{A}$ loses in the $I\text{-}EU\text{-}CMA$ experiment then $\mathcal{B}$ loses in the $EU\text{-}CMA$ and AE experiments for $ASig$, and $\mathcal{B}$ *aborts*. Otherwise, given $m^* = (m_1^*, \ldots, m_k^*), 1 \le k \le u$ and $\gamma^* = (\sigma^*, r^*)$, $\mathcal{B}$ sets the forgery (or aggregate extraction) as $(\widetilde{m}^*, \sigma^*)$, where $\widetilde{m}^* = (m^*, r^*)$. $\mathcal{B}$ continues as follows:

a) *Forgery*: If $m^* \not\subseteq \{\overrightarrow{m}_j\}_{j=1}^{q_s}$ or $r^* \notin \mathcal{LR}$ holds then $\mathcal{B}$ returns the forgery $(\widetilde{m}^*, \sigma^*)$ against $ASig.Sign$. This forgery is non-trivial since $\widetilde{m}^* \notin \mathcal{LD}$. The forgery is valid since $PISB\text{-}RP.Ver(PK, m^*, \gamma^*) = 1$ implies $ASig.Ver(PK, \widetilde{m}^*, \sigma^*) = 1$ and $|r| = \kappa$.
b) *Aggregate Extraction*: If $\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq ||_{k \in I} \overrightarrow{m}_k$ and $r^* \in \mathcal{LR}$ hold then it implies the following conditions also hold as in Definition 8:
   (i) $ASig.Ver(PK, \widetilde{m}^*, \sigma^*) = 1$,
   (ii) $\exists I' \subseteq \{1, \ldots, q_s\} : \widetilde{m}^* \subseteq ||_{k \in I'} \widetilde{m}_k$, $\widetilde{m}_k \in \mathcal{LD}$,
   (iii) $\forall \overline{I} \subseteq \{1, \ldots, q_s\} : [\widetilde{m}^* || (||_{j \in \overline{I}} \widetilde{m}_j)] \neq \{\widetilde{m}_l\}_{l=1}^{q_s}$, since $r^* \in \mathcal{LR}$ and $r^* \neq \{\widetilde{m}_l \in \mathcal{LD}\}_{l=1}^{q_s}$ hold.

   Since the validity and non-triviality conditions described in Definition 8 are satisfied, $\mathcal{B}$ returns $(\widetilde{m}^*, \sigma^*)$ as an aggregate extraction against $ASig.Sign$.

The execution time and probability analysis are similar to Theorem 1 (i.e., the simulation is perfectly indistinguishable).

6 Performance Analysis

We now present the performance analysis of $PISB$ schemes and compare them with the existing alternatives.

6.1 Computational Overhead

In $PISB\text{-}Generic$, the server requires a $Sig.Sign$ plus the aggregation of l individual $ASig$ signatures. The client requires a $Sig.Ver$ plus $ASig.Ver$ for l data items. In $PISB\text{-}CSA\text{-}RSA$, the server requires a $C\text{-}RSA.Sig$ computation plus l modular multiplications. The client requires a single $RSA.Ver$ plus $C\text{-}RSA.Ver$ for l data items.

The $PISB\text{-}CSA\text{-}RSA$ is the most *client efficient scheme* among all of its counterparts, since it only requires RSA and $C\text{-}RSA$ signature verifications with a small exponent (e.g., $e = 3$). Therefore, it is an ideal choice for battery or computational limited queriers such as mobile devices. However, it requires a full RSA exponentiation at the server side. Notice that even with the server side exponentiation, the end-to-end delay of $PISB\text{-}CSA\text{-}RSA$ is *50, 65, 41, and 55 times lower* than

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

that of $PISB\text{-}Generic$, GQ-based, SKROOT-based and iBGLS schemes, respectively. The end-to-end delay of $PISB\text{-}RP$ instantiated with $C\text{-}RSA$ is $(l+1)H + (2l+4) \cdot Mul$. This is the *lowest computational end-to-end delay* among all compared alternatives being a magnitude(s) of times more efficient than previous schemes [7], [12] as well as other $PISB$ variants (see Table 2 and Figure 2d for execution time comparisons without and with simulated network delays, respectively). Finally, $PISB\text{-}Generic$ can be instantiated with various signature schemes, which allow different performance trade-offs (see Table 1).

6.2 Communication and Storage Overhead

$PISB$ schemes do not require multi-round communication to achieve the immutability. Therefore, their signature overhead is the aggregate signature plus the protection signature in $PISB\text{-}Generic$, and only the aggregate signature itself in $PISB\text{-}CSA\text{-}RSA$. The private and public key sizes are the sum of that of their base signature schemes.

$PISB\text{-}Generic$ with BGLS and ECDSA has the smallest key and signature sizes among its counterparts with the exception of iBGLS (which has a much larger end-to-end delay and client computation as seen in Table 2). $PISB\text{-}CSA\text{-}RSA$ also has much smaller signature and key sizes than that of GQ-based and SKROOT-based schemes. Despite GQ-based scheme is client and server computationally efficient, it is not practical due to its multi-round communication (introduces a substantial communication delay as shown in Figure 2d). Multi-round communication is undesirable for wireless and low bandwidth applications due to the packet loss potential.

$PISB\text{-}RP$ requires transmitting κ-bit (e.g., 80 bits) random number along with the aggregate signature. It also requires storing $(|\sigma| + \kappa)$ pre-computed token-signature pairs per query as in other pre-computation methods (e.g., [23], [25], [27]). These signature size and pre-storage overhead are more efficient than that of $PISB\text{-}Generic$ instantiation with ECDSA token and offline/online signatures (e.g., [23], [25], [27]) as shown in Table 1 and Table 2. However, notice that, all pre-computation methods including $PISB\text{-}RP$ increases the server storage compared to $PISB$ schemes and previous schemes (e.g., [7], [12]) that do not rely on pre-computation.

6.3 Discussions

Table 1 and Table 2 compare our schemes with previous schemes. We give details of the comparison as follows:

• *Instantiations*: In $PISB\text{-}Generic$ instantiated with $C\text{-}RSA$ and *OO2*, we assume that one-time public key for per-item is transmitted along with the one-time signature itself. Another possibility is to pre-deploy a one-time public keys to the client side, which would then incur a linear public key size (and also pre-deployment still requires the transmission of a one-time public keys). One may replace HORS [27] with some variants such as TV-HORS [29] that does not require transmitting or pre-storing one-time public keys via public key hash chaining methods. However, such methods sacrifice security (i.e., time-valid security), introduce non-negligible packet loss and requires a loose-time synchronization [29]. In $PISB\text{-}RP$ instantiated with $C\text{-}RSA$, the size of private/public key are $|n|$ but not $2|n|$, since both the individual signatures to be aggregated and random number-signature pairs are generated by the data owner with the same private key.

We select some instantiations from Table 1 to provide representative numeric values in Table 2. We select $PISB\text{-}Generic$ instantiated with BGLS, since it provides the smallest signature and private/public key sizes (without pre-computation) and also enables the multiple-signer aggregation. $PISB\text{-}CSA\text{-}RSA$ is a versatile scheme with high computational efficiency and plausible signature/key sizes without requiring pre-computation. We select $PISB\text{-}RP$ instantiated with $C\text{-}RSA$, since it achieves the highest computational efficiency, low communication and end-to-end delay overall, but it requires pre-storage. Notice that $PISB\text{-}RP$ has a smaller pre-storage overhead than that among other pre-computation alternatives with similar end-to-end delay (e.g., *(CRSA,OO1)* and *(CRSA,OO2)* as seen in Table 1, but *(BGLS,ECDSA-p)* has a high end-to-end delay due to BGLS). $PISB\text{-}Generic$ is instantiated with BGLS [8] as $ASig$ (20 byte) and with ECDSA [55] as Sig (40 byte protection signature) for pre-computed parameters (0.36 ms) [59] or pre-computed tokens [24] (0.03 ms). $PISB\text{-}RP$ in this example is instantiated with $C\text{-}RSA$. $PISB\text{-}RP$ generically can achieve both multiple (e.g., via BGLS instantiation) and single signer aggregation.

The immutable signature size is the aggregate signature size plus the size of additional cryptographic tags transmitted (e.g., protection signatures, values transmitted for multi-rounds).

• *Parameters and Measurements*: Given $\kappa = 80$, we select $|n| = 1024$, $|H| = 160$, $|q'| = 160$, $|p'| = 512$, $|b| = 30$, $z = 20$, $t = 256$. The estimated execution times are measured for $l = 10$ data items (query elements). Estimated execution times are measured on a computer with an Intel(R) Core(TM) i7 Q720 at 1.60GHz CPU and 2GB RAM running Ubuntu 10.10. We used MIRACL [59] library for all the measurements including cryptographic pairing operations.

• *Multi-rounds and Delays*: GQ-based scheme needs three communication rounds (each three passes) to achieve $\kappa \geq 80$, in which each pass needs to transmit an element from Z_N^*. End-to-end delay is the sum of client and server execution times plus the estimated communication delay introduced by multi-rounds. Only GQ-based scheme requires multi-rounds, which substantially increase its end-to-end delay. We only consider the delay introduced due to extra rounds (but not the unavoidable one-way transmission from server to the client, which exists in all compared schemes). The average network delay is assumed as 30 ms round trip time for a single query.

• *Pre-storage overhead*: $PISB\text{-}RP$, offline/online signatures (e.g., [23], [25], [26] and $PISB\text{-}Generic$ instantiations that rely offline/online signatures and ECDSA tokens [24] require storing a pre-computed token in order to generate a signatures during the online phase (see the last column of Table 1). For instance, given $a = 10^4$ queries to be processed, $PISB\text{-}RP$ with $C\text{-}RSA$, $PISB\text{-}Generic$ with *(BGLS,ECDSA-p), (CRSA,OO1)* and *(CRSA,OO2)* require approximately 1.3 MB, 0.5 MB, 48 MB and 2.5 MB pre-storage, respectively.

• *Figures*: Figure 2a and Figure 2b compare estimated execution times of $PISB$ and previous schemes for the client and server sides, respectively. These estimated execution times are based on the values presented in Table 2 and are projected for the increasing numbers of queries up to $a = 10^4$ (for $l = 10$ data items for each query). Figure 2c compares the communication overhead of $PISB$ and previous schemes. These costs are based on only signature sizes but not the transmitted data items (which are the same for all compared schemes). Figure 2d compares end-to-end delay of $PISB$ and previous schemes by considering average

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

TABLE 1: The client and server overhead of $PISB$ and its counterparts for l data items (analytical)

-		Client Comp.	Server Comp.	Sig.	SK	PK	Pre-store
$PISB$-Generic	***Generic*, multiple**	$Sig.Ver + ASig.Ver^{l}$	$Sig.Sign + ASig.Agg^{l}$	$\|\sigma\|$	$u\|sk\|$	$u\|pk\|$	-
	(BGLS,ECDSA-P)	$\boldsymbol{l(BM + H) + 1.3 \cdot Emul}$	$\boldsymbol{l \cdot EAdd}$	$\boldsymbol{3\|q'\|}$	$\boldsymbol{2\|q'\|}$	$\boldsymbol{2(\|p'\| + \|q'\|)}$	$a(2\|q'\| + \kappa)$
	(CRSA,OO1)	$\boldsymbol{l(H + Mul) + Exp\|_{\|n\|}^{\|n\|}}$	$\boldsymbol{l \cdot Mul + H}$	$\boldsymbol{3\|n\|}$	$\boldsymbol{3\|n\|}$	$\boldsymbol{3\|n\|}$	$a(2\|n\|)$
	(CRSA,OO2)	$\boldsymbol{l(H + Mul) + z \cdot H}$	$\boldsymbol{l \cdot Mul}$	$(z + t)(\|H\| + \|n\|)$	$t \cdot (\|H\| + \|n\|)$	$t \cdot (\|H\| + \|n\|)$	$a(t \cdot \|H\|)$
$PISB$-RP	***Generic*, multiple**	$ASig.Ver^{l+1}$	$\boldsymbol{ASig.Agg^{l+1}}$	$\boldsymbol{\|\sigma\|}$	$\boldsymbol{u\|sk\|}$	$\boldsymbol{u\|pk\|}$	$a(\|\sigma\| + \kappa)$
	***BGLS*, $u = 2$**	$l \cdot (BM + H)$	$\boldsymbol{l \cdot EAdd}$	$\boldsymbol{\|q'\|}$	$\boldsymbol{2\|q'\|}$	$\boldsymbol{2(\|p'\| + \|q'\|)}$	$a(\|q'\| + \kappa)$
	CRSA	$l \cdot (H + Mul)$	$\boldsymbol{l \cdot Mul}$	$\boldsymbol{\|n\| + \kappa}$	$\boldsymbol{\|n\|}$	$\boldsymbol{\|n\|}$	$a(\|n\| + \kappa)$
PISB-CSA-RSA		$\boldsymbol{l(H + Mul)}$	$\boldsymbol{Exp\|_{\|n\|}^{\|n\|} + l \cdot Mul}$	$\boldsymbol{\|n\|}$	$\boldsymbol{2\|n\|}$	$\boldsymbol{2\|n\|}$	-
GQ-based [7]		$3Exp\|_{\|n\|}^{\|b\|} + l(H + Mul)$	$3Exp\|_{\|n\|}^{\|b\|} + l \cdot Mul$	$9\|n\|$	$\|b\| + \|n\|$	$\|b\| + \|n\|$	-
SKROOT-based [7]		$4Exp\|_{\|2n\|}^{\|n\|} + l(H + Mul)$	$4Exp\|_{\|2n\|}^{\|n\|} + l \cdot Mul$	$4\|n\|$	$5\|n\|$	$\|n\|$	-
iBGLS [7]		$l \cdot (BM + H)$	$EMul + l \cdot EAdd$	$\|q'\|$	$\|q'\|$	$\|p'\| + \|q'\|$	-

• *Notation*: We denote our proposed schemes and the metrics that they outperform at least one of the compared schemes in bold. $Exp|_{|y|}^{|x|}$ denotes a modular exponentiation with a modulus and exponent sizes $|y|$ and $|x|$, respectively. Mul denotes modular multiplication under modulus n. BM, $EAdd$, and $EMul$ denote ECC bilinear map, scalar addition and scalar multiplication over modulus q', respectively. $Mulq'$ denotes modular multiplication under modulus q'. $ASig.Ver^{l}$ denotes the aggregate signature verification for l items (the notation applies to $ASig.Agg$). We omit constant number of low-cost operations if there is an expensive operation (e.g., a single H or Mul is omitted if there is an Exp). We use double-point scalar multiplication for ECDSA verifications ($1.3 \cdot Emul$ instead of $2 \cdot EMul$). $|H|$, $|\sigma|$, $|sk|$ and $|pk|$ denote the bit lengths of the output of H, signature, private key and public key for Sig, respectively (Sig is selected as ECDSA with $|q'|$ in Table 1). *OO1* and *OO2* refer to Shamir's offline/online scheme [25] and generic offline/online scheme [26] instantiated with HORS [27], respectively. *(BGLS,ECDSA-P)* means $PISB$-$Generic$ is instantiated with BGLS as the aggregate signature and ECDSA with token Pre-computation [24] as the protection signature. The similar notation applies to *(CRSA,OO1)* and *(CRSA,OO2)*. For $PISB$ schemes "multiple" denoted in generic scheme means multiple-signer for u distinct data owners. All other $PISB$ schemes are evaluated in single signer mode. Variables a denotes the number of queries to be made by the client (also implies the number of pre-computed tokens to be stored by the server) in a given session. Variables (z, t) are parameters used in HORS [27].

TABLE 2: The client/server overhead of $PISB$ and its counterparts for $l = 10$ items in a query (in ms)

-	$PISB$-$Generic$	$PISB$-CSA-RSA	$PISB$-RP	GQ-based [7]	SKROOT [7]	iBGLS [7]
Server Comp.	0.66 / 0.39	4.03	0.22	1.5	92.4	2.2
Client Comp.	224.97	0.46	0.41	1.57	92.77	245.7
Extra rounds	0	0	0	3 rounds (each 3 passes)	0	0
(Est.) End-to-end	225.63	4.49	0.63	292	185.17	247.9
Signature size	60 byte	1 KB	1.1KB	9 KB	4 KB	20 byte
sk size	40 byte	2 KB	1KB	1 KB	5 KB	20 byte
pk size	80 byte	1 KB	1 KB	1 KB	1 KB	40 byte
Pre-store (server), (per-message)	50 byte	-	138 byte	-	-	-
Pre-computation cost (offline, per-message)	0.68	-	4.06	-	-	-
Aggregation Type	Multiple/single	Multiple/single	Single	Single	Single	Multiple
Provable Sec.	Yes	Yes	Yes	No	No	No
Special Pre-computation	Yes	Yes	Yes	No	No	No

execution time plus simulated network delays. The estimated end-to-end delay is the server and client execution times plus the simulated network delay. We assume that the network delay for each query varies between 20 ms to 40 ms for a round-trip communication (fluctuations in the plot are caused by these varying delays). Notice that interactive schemes (e.g., GQ in [7], [12]) are negatively affected from network delays.

These comparisons show that $PISB$ schemes are the most computational and communication efficient schemes with the lowest end-to-end delay among all compared alternatives.

7 Conclusion

In this paper, we developed new cryptographic schemes called $PISB$, which provide practical immutable signatures for outsourced databases. We also gave the first formal security assessment of immutable signatures for outsourced databases, highlighted the relationship between aggregate extraction problem and signature immutability, and then provided formal proofs for $PISB$ schemes. We also demonstrated that $PISB$ schemes are much more efficient than previous immutable signatures: $PISB$-$Generic$ describes a simple yet efficient way to obtain immutable constructions via standard signatures that is more efficient than previous solutions. $PISB$-CSA-RSA offers a very low verifier computational overhead and high communication efficiency via a C-RSA based construction, which is ideal for battery/computation limited queriers (e.g., mobile devices). $PISB$-RP achieves the lowest end-to-end delay among all compared alternatives via special pre-computation techniques, which is desirable for task-intensive applications by increasing the service quality. All $PISB$ schemes are non-interactive and have small signature sizes. Hence, $PISB$ schemes are ideal choices for providing immutability, authentication and integrity services for outsourced database systems.

References

[1] H. Hacigumus, B. Iyer, and S. Mehrotra, "Providing database as a service," in *Proceedings of the 18th International Conference on Data Engineering*, ser. ICDE '02, Washington, DC, USA, 2002, pp. 29–38.

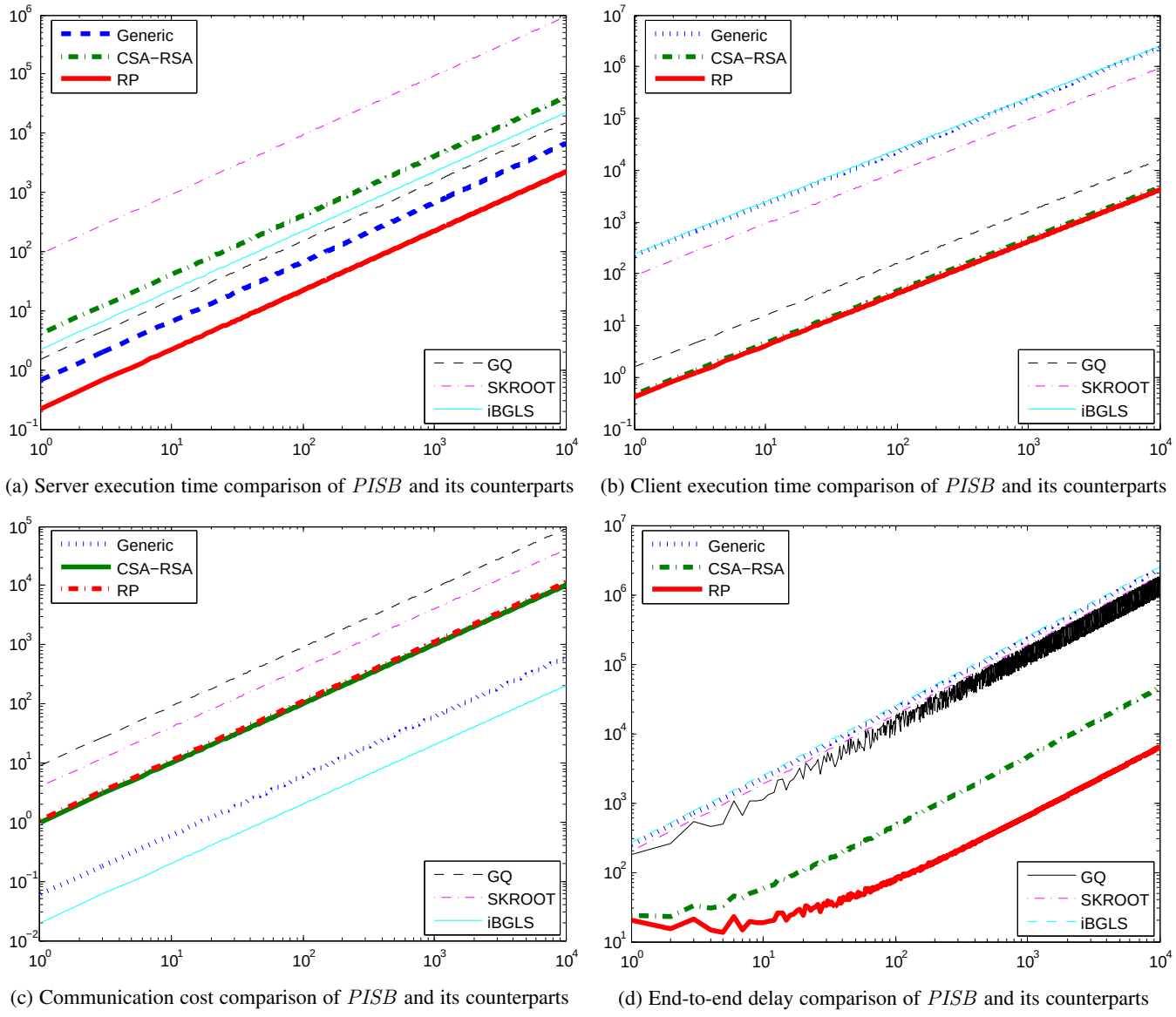
[2] R. S., "Secure data outsourcing," in *Proceedings of the 33rd international conference on Very large data bases (VLDB)*, 2007, pp. 1431–1432.

[3] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *Transaction on Storage (TOS)*, vol. 2, no. 2, pp. 107–138, 2006.

[4] A. Patel, S. J. Nirmala, and S. M. Bhanu, "Security and availability of data in the cloud," in *Advances in Computing and Information Technology*, ser. Advances in Intelligent Systems and Computing. Springer Berlin Heidelberg, 2012, vol. 176, pp. 255–261.

[5] H. Wang and L. V. S. Lakshmanan, "Efficient secure query evaluation over encrypted xml databases," in *Proceedings of the 32nd international conference on Very large data bases*, ser. VLDB '06, 2006, pp. 127–138.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing



(a) Server execution time comparison of $PISB$ and its counterparts

(b) Client execution time comparison of $PISB$ and its counterparts

(c) Communication cost comparison of $PISB$ and its counterparts

(d) End-to-end delay comparison of $PISB$ and its counterparts

Fig. 2: Computation, communication and end-to-end delay comparison of $PISB$ schemes and their counterparts. $PISB$-$Generic$ and $PISB$-RP are instantiated with ($BGLS$,$ECDSA$-P) and C-RSA. Figures are in log-log (base 10) scale.

[6] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, "Privacy-preserving group data access via stateless oblivious ram simulation," in *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012, pp. 157–167.

[7] E. Mykletun, M. Narasimha, and G. Tsudik, "Signature bouquets: Immutability for aggregated/condensed signatures," in *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS '04)*. Springer-Verlag, September 2004, pp. 160–176.

[8] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. of the 22th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '03)*. Springer-Verlag, 2003, pp. 416–432.

[9] A. A. Yavuz, "Practical immutable signature bouquets (PISB) for authentication and integrity in outsourced databases," in *27th Annual on Data and Applications Security and Privacy*, ser. DBSec '13, July 15-17 2013.

[10] J. Coron and D. Naccache, "Boneh et al.'s k-element aggregate extraction assumption is equivalent to the diffie-hellman assumption," in *Proceedings of the 9th International Conference on the Theory and Application of Cryptology (ASIACRYPT 03')*, 2003, pp. 392–397.

[11] R. Johnson, D. Molnar, D. X. Song, and D. Wagner, "Homomorphic signature schemes," in *CT-RSA*, 2002, pp. 244–262.

[12] E. Mykletun and G. Tsudik, "Aggregation queries in the database-as-a-service model," in *Proceedings of the 20th IFIP WG 11.3 working conference on Data and Applications Security*, ser. DBSEC'06. Springer-Verlag, 2006, pp. 89–103.

[13] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracles," in *Proc. of the 25th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '06)*. Springer-Verlag, 2006, pp. 465–485.

[14] A. Boldyreva, C. Gentry, A. O'Neill, and D. Yum, "Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing," in *Proceedings of the 14th ACM Conference on Computer and Communications Security, (CCS '07)*. ACM, 2007, pp. 276–285.

[15] H. Zhu and J. Zhou, "Finding compact reliable broadcast in unknown fixed-identity networks (short paper)," in *Proc. of the 6th International Conference on Information and Communications Security (ICICS '06)*, 2006, pp. 72–81.

[16] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham, "Sequential aggregate signatures from trapdoor permutations," in *Proc. of the 23th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '04)*. Springer-Verlag, 2004, pp. 74–90.

[17] L. C. Guillou and J. J. Quisquater, "A "paradoxical" identity-based signature scheme resulting from zero-knowledge," in *Proceedings on Advances in Cryptology (CRYPTO '88)*. Springer-Verlag, 1988, pp.

216–231.

[18] P. Samarati and S. D. C. di Vimercati, "Data protection in outsourcing scenarios: issues and directions," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10, 2010, pp. 1–14.

[19] B. Thompson, S. Haber, W. G. Horne, T. Sander, and D. Yao, "Privacy-preserving computation and verification of aggregate queries on outsourced databases," in *Proc. of the 9th Int. Symposium on Privacy Enhancing Technologies*, ser. PETS '09, 2009, pp. 185–201.

[20] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Transaction on Storage (TOS)*, vol. 5, no. 1, pp. 1–21, 2009.

[21] A. A. Yavuz, P. Ning, and M. K. Reiter, "BAF and FI-BAF: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems," *ACM Transaction on Information System Security*, vol. 15, no. 2, 2012.

[22] A. Yavuz, P. Ning, and M. Reiter, "Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging," in *Financial Cryptography and Data Security (FC 2012)*, ser. Lecture Notes in Computer Science, March 2012, vol. 7397, pp. 148–163.

[23] S. Even, O. Goldreich, and S. Micali, "Online/offline digital signatures," in *Proceedings on Advances in Cryptology (CRYPTO '89)*. Springer-Verlag, 1989, pp. 263–275.

[24] D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli, "Can D.S.A. be improved? Complexity trade-offs with the digital signature standard," in *Proceedings of the 13th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '94)*, 1994, pp. 77–85.

[25] A. Shamir and Y. Tauman, "Improved online/offline signature schemes," in *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '01. London, UK: Springer-Verlag, 2001, pp. 355–367.

[26] D. Catalano, M. D. Raimondo, D. Fiore, and R. Gennaro, "Off-line/on-line signatures: Theoretical aspects and experimental results," ser. Public Key Cryptography (PKC). Springer-Verlag, 2008, pp. 101–120.

[27] L. Reyzin and N. Reyzin, "Better than BiBa: Short one-time signatures with fast signing and verifying," in *Proceedings of the 7th Australian Conference on Information Security and Privacy (ACIPS '02)*. Springer-Verlag, 2002, pp. 144–153.

[28] J. Lee, S. Kim, Y. Cho, Y. Chung, and Y. Park, "HORSIC: An efficient one-time signature scheme for wireless sensor networks," *Information Processing Letters*, vol. 112, no. 20, pp. 783 – 787, 2012.

[29] Q. Wang, H. Khurana, Y. Huang, and K. Nahrstedt, "Time valid one-time signature for time-critical multicast data authentication," in *INFOCOM 2009, IEEE*, April 2009.

[30] J. Pieprzyk, H. Wang, and C. Xing, "Multiple-time signature schemes against adaptive chosen message attacks," in *Selected Areas in Cryptography (SAC)*, 2003, pp. 88–100.

[31] W. Neumann, "HORSE: An extension of an r-time signature scheme with fast signing and verification," in *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, vol. 1, april 2004, pp. 129 – 134 Vol.1.

[32] A. A. Yavuz, "Eta: efficient and tiny and authentication for heterogeneous wireless systems," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, ser. WiSec '13. New York, NY, USA: ACM, 2013, pp. 67–72.

[33] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[34] P. Horster, H. Petersen, and M. Michels, "Meta-elgamal signature schemes," in *Proceedings of the 2nd ACM Conference on Computer and communications security*, ser. CCS '94. New York, NY, USA: ACM, 1994, pp. 96–107.

[35] A. Yavuz, "An efficient real-time broadcast authentication scheme for command and control messages," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 10, pp. 1733–1742, Oct 2014.

[36] R. Tamassia, "Authenticated data structures," in *Algorithms - ESA 2003*, ser. Lecture Notes in Computer Science, G. Di Battista and U. Zwick, Eds. Springer Berlin Heidelberg, 2003, vol. 2832, pp. 2–5.

[37] C. Papamanthou and R. Tamassia, "Time and space efficient algorithms for two-party authenticated data structures," in *Information and Communications Security*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, vol. 4861, pp. 1–15.

[38] M. Etemad and A. KÃijpÃğÃij, "Database outsourcing with hierarchical authenticated data structures," in *Information Security and Cryptology – ICISC 2013*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 381–399.

[39] J. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, ser. EUROCRYPT '93. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1994, pp. 274–285.

[40] M. Goodrich, R. Tamassia, and J. Hasic, "An efficient dynamic and distributed cryptographic accumulator," in *Information Security*, ser. Lecture Notes in Computer Science, 2002, vol. 2433, pp. 372–388.

[41] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Authenticated hash tables based on cryptographic accumulators," *Algorithmica*, pp. 1–49, 2015.

[42] R. Merkle, "Protocols for public key cryptosystems," in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Apr 1980.

[43] H. Pang and K.-L. Tan, "Authenticating query results in edge computing," in *Data Engineering, 2004. Proceedings. 20th International Conference on*, March 2004, pp. 560–571.

[44] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis, "Authenticated join processing in outsourced databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 5–18.

[45] W. Pugh, "Skip lists: A probabilistic alternative to balanced trees," *Commun. ACM*, vol. 33, no. 6, pp. 668–676, Jun. 1990.

[46] J. Wang and X. Du, "Skip list based authenticated data structure in das paradigm," in *Grid and Cooperative Computing, 2009. GCC '09. Eighth International Conference on*, Aug 2009, pp. 69–75.

[47] B. Palazzi, M. Pizzonia, and S. Pucacco, "Query racing: Fast completeness certification of query results," in *Data and Applications Security and Privacy XXIV*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6166, pp. 177–192.

[48] S. Crosby and D. S. Wallach, "Efficient data structures for tamper evident logging," in *Proceedings of the 18th conference on USENIX Security Symposium*, August 2009.

[49] R. Ostrovsky and W. E. Skeith, III., "A survey of single-database private information retrieval: techniques and applications," in *Proc. of the 10th international conference on Practice and theory in public-key cryptography*, ser. PKC'07. Springer-Verlag, 2007, pp. 393–411.

[50] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and Communications Security (CCS '93)*. NY, USA: ACM, 1993, pp. 62–73.

[51] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

[52] M. Bellare and P. Rogaway, "The exact security of digital signatures: How to sign with RSA and Rabin," in *Proceedings of the 15th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '96)*. Springer-Verlag, 1996, pp. 399–416.

[53] A. Joux and K. Nguyen, "Separating decision diffie-hellman from computational diffie-hellman in cryptographic groups," *Journal of Cryptology*, vol. 16, no. 4, pp. 239–247, 2003.

[54] A. A. Yavuz and P. Ning, "BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems," in *Proceedings of 25th Annual Computer Security Applications Conference (ACSAC '09)*, 2009, pp. 219–228.

[55] *ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, American Bankers Association, 1999.

[56] X. Ding and G. Tsudik, "Simple identity-based cryptography with mediated rsa," in *Proceedings of the RSA conference on the cryptographers' track (CT-RSA '03)*. Springer-Verlag, 2003, pp. 193–210.

[57] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[58] D. Stinson, *Cryptography: Theory and Practice,Second Edition*. CRC/C&H, 2002.

[59] Shamus, "Multiprecision integer and rational arithmetic c/c++ library (MIRACL)," http://www.certivox.com/miracl/miracl-download/, Last Accessed on 09/02/2014.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2530708, IEEE Transactions on Dependable and Secure Computing

Attila Altay Yavuz received a BS degree in Computer Engineering from Yildiz Technical University (2004) and a MS degree in Computer Science from Bogazici University (2006), both in Istanbul, Turkey. He received his PhD degree in Computer Science from North Carolina State University in August 2011. Between December 2011 and July 2014, he was a member of the security and privacy research group at the Robert Bosch Research and Technology Center North America. Since August 2014, he has been an Assistant Professor in the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, USA. He is also an adjunct faculty at the University of Pittsburgh's School of Information Sciences since January 2013.

Attila Altay Yavuz is interested in design, analysis and application of cryptographic tools and protocols to enhance the security of computer networks and systems. His current research focuses on the following topics: Privacy enhancing technologies (e.g., dynamic symmetric and public key based searchable encryption), security in cloud computing, authentication and integrity mechanisms for resource-constrained devices and large-distributed systems, efficient cryptographic protocols for wireless sensor networks.