Practical Immutable Signature Bouquets (PISB) for Authentication and Integrity in Outsourced Databases

Attila A. Yavuz

Robert Bosch LLC, Research and Technology Center - North America
2835 East Carson Street, Pittsburgh, PA 15203
attila.yavuz@us.bosch.com

Abstract. Database outsourcing is a prominent trend that enables organizations to offload their data management overhead (e.g., query handling) 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 and communication costly (e.g., highly interactive), which make them impractical for task-intensive and heterogeneous applications.

In this paper, we developed two new schemes that we call *Practical and Immutable Signature Bouquets (PISB)*, which achieve efficient immutability for outsourced database systems. Both $PISB$ schemes are very simple, non-interactive, and computation/communication efficient. Our generic scheme can be constructed from any aggregate signature coupled with a standard signature. Hence, it can flexibly provide performance trade-offs for various types of applications. Our specific scheme is constructed from Condensed-RSA and Sequential Aggregate RSA. It has a very low verifier computational overhead and end-to-end delay with a small signature size. We showed that $PISB$ schemes are secure and also much more efficient than previous alternatives.

Keywords: Applied cryptography, outsourced databases, immutable digital signatures, distributed systems, public key cryptography.

1 Introduction

It is a growing trend that the data is outsourced and being managed (e.g., query handling, maintenance) 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 and access for the hosted data [2].

[1] http://www.ibm.com/software/data/db2
[2] http://www-935.ibm.com/services/us/en/it-services/storage-and-data-services.html

L. Wang and B. Shafiq (Eds.): DBSec 2013, LNCS 7964, pp. 179–194, 2013.

Data outsourcing can significantly reduce the cost of data management (e.g., via continuous service, expertise, upgrade/maintanence) 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 or fully) 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.

In this paper, we focus on providing 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.

In Section 1.1, we give our data and system models. In Section 1.2, we describe the signature mutability problem and limitations of existing solutions. In Section 1.3, we summarize our contributions and highlight the desirable properties of our schemes.

1.1 System and Data Model

We follow Mykletun et al.'s Outsourced Database Model (ODB) [3,7], which is a variant of traditional "database as a service" model [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 with these services, but it is *not* trusted with the integrity and authentication of the hosted 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.

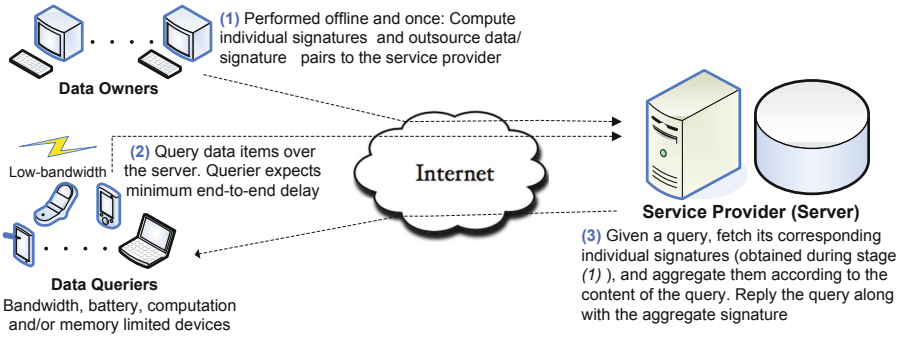

Fig. 1. Mykletun et al.'s Outsourced Database Model (ODB)

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 mobile device or a PDA). A querier can make a query on the database elements belonging to a single or multiple data owners. The former is called *non-cross signer queries* while the latter is called *cross signer queries*. The data querier verifies the corresponding aggregate signature of her query, along with cryptographic tokens transmitted for the immutability.

Figure 1 summarizes the ODB model described above.

Data Model and Scope: We assume that the data is managed with a traditional relational database management system and the queries are formulated with SQL. Our work handles SQL queries involving SELECT clauses, which return the selection of a set of records or fields matching a given predicate. Our work does *not* address SQL queries involving data aggregation that return a single value for a given query.

The granularity of data integrity and authentication may vary according to the application (e.g., attribute level). 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 and Limitations of Existing Solutions

Ability to aggregate different signatures into a single signature is advantageous, but it also allows any party to derive new aggregate signatures from the existing ones. For instance, assume that the server provides signatures $(\sigma_{1,l}, \sigma_{1,k})$ on queries $(m_1, \ldots, m_l)$ and $(m_1, \ldots, m_k)$, respectively, where $1 < k < l$ and the aggregation operation is addition (e.g., [8]). Any querier can derive a valid signature on query elements $m_{k+1}, \ldots, m_l$ (that have *not* been queried before) by simply computing $\sigma_{k+1,l} = \sigma_{1,l} - \sigma_{1,k}$.

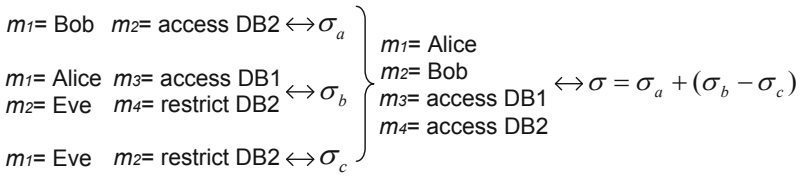

Fig. 2. An example of signature mutability in the content access control applications

This property has undesirable effects on many real-life applications. One 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). A group of clients can possess different access privileges from those owned by each client in isolation. Figure 2 exemplifies how three colluding clients can derive a valid token (i.e., signature) with an access right beyond their actual privileges.

Another example is *paid database services*, in which the server acts as an authorized re-distribution agent for the information contained in the outsourced database. Assume that the server charges a fee for each music album queried (downloaded) over the outsourced database. Signature mutability allows an unauthorized splitting and re-distribution of authentic query replies. Colluding clients may gather various music albums and their signatures, and combine and re-sell them according to their will (without paying/obtaining any authorization) [7].

Mykletun et al. [7] introduce signature immutability techniques to address these problems. Their RSA-based techniques prevent an adversary from deriving new signatures by hiding the actual aggregate signature via an interactive Guillou-Quisquater (GQ) [9] 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 BLS 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.

1.3 Our Contribution

To address these limitations, we develop novel cryptographic schemes called *Practical Immutable Signature Bouquets (PISB)*, which is suitable for outsourced database systems. Specifically, we developed a Condensed-RSA (C-RSA) and Sequential Aggregate RSA (SA-RSA) based scheme called $PISB\text{-}CSA\text{-}RSA$ and a generic scheme called $PISB\text{-}Generic$. We summarize the desirable properties of our schemes below:

- *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 $PISB\text{-}CSA\text{-}RSA$

Table 1. The client/server overhead of $PISB$ and its counterparts for 10 items in a query (in ms)

-	$PISB$-$Generic$	$PISB$-CSA-RSA	GQ-based [7]	SKROOT [7]	iBGLS [7]
Server Comp.	0.66 / 0.39	4.03	1.5	92.4	2.2
Client Comp.	224.97	0.46	1.57	92.77	245.7
Extra rounds	0	0	3 rounds (each 3 passes)	0	0
(Est.) End-to-end	225.63	4.49	292	185.17	247.9
Signature size	60 byte	1 KB	9 KB	4 KB	20 byte
sk size	40 byte	2 KB	1 KB	5 KB	20 byte
pk size	80 byte	1 KB	1 KB	1 KB	40 byte
Aggregation Type	Cross/non-cross	Non-cross	Non-cross	Non-cross	Cross
Provable Sec.	Yes	Yes	No	No	No
Pre-computability	Yes	No	No	No	No

• Analytical comparison, key/parameter sizes and measurement details are given in Section 6.
• 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).
• $PISB$-$Generic$ is instantiated with BLS [8] as $ASig$ (20 byte aggregate signature) and with ECDSA [10] as Sig (40 byte protection signature) for pre-computed parameters (0.36 ms) [11] or pre-computed tokens [12] (0.03 ms).
• 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.
• *Remark*: $PISB$-CSA-RSA is significantly more efficient than all of its counterparts at the client side, which makes it suitable for mobile or resource-constrained queriers. Its end-to-end delay is also *40 to 60 times lower than its alternatives*. $PISB$-$Generic$ offers various performance trade-offs with its generic structure (e.g., only alternative with pre-computability). This instantiation of $PISB$-$Generic$ offers small signature/key sizes and high server efficiency simultaneously. $PISB$-$Generic$ and $PISB$-CSA-RSA are suitable for cross and non-cross signer models, respectively (see Section 1.2). $PISB$ schemes are also provable secure.

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).

- *High Computational Efficiency*: $PISB$ schemes are much more computationally efficient than their counterparts. $PISB$-CSA-RSA is the most client efficient scheme among its counterparts, 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 (queriers) such as mobile and hand-held devices. It is also plausibly efficient at the server side while achieving this client efficiency. $PISB$-$Generic$ is the most server efficient scheme among its counterparts due to its pre-computability property. This enables the server to responde large number queries in peak times without being bottlenecked.
- *Small Signature Sizes*: $PISB$-CSA-RSA is the only RSA-based scheme that can compute a compact immutable aggregate signature, which makes it more communication efficient than its 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).
- *Minimum End-to-end Delay*: $PISB$ schemes have a much smaller end-to-end delay than all of their counterparts, which offers a better service quality.

- *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.

Table 1 outlines the properties and compares $PISB$ schemes with their counterparts.

The remainder of this paper is organized as follows. Section 2 provides definitions used by our schemes. Section 3 defines our security model. Section 4 describes the proposed schemes in detail. Section 5 provides the security analysis. Section 6 gives the performance analysis and compares our schemes with their counterparts. Section 7 outlines related work and Section 8 concludes this paper.

2 Preliminaries

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

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)* [13], 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 in total. 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 *cross 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* σ *as the output.*
- $c \leftarrow ASig.Ver(\{pk_i, m_i\}_{i=1}^u, \sigma_{1,u})$*: The aggregate 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 $EU\text{-}CMA$ experiment for $ASig$ is a straightforward extension of Definition 2, in which $\mathcal{A}$ is required to produce a forgery under a public key $pk \in \overrightarrow{pk}$ that is not under his control during the experiment (see [8] for details).

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

Definition 4. $C\text{-}RSA$ *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* $(e, d) \in Z^*_{\phi(n)}$ *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., [14]) 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., [15]) requires that the signature generation and verification are performed in a specific order. The signature generation and aggregation operations are unified.

In $PISB\text{-}CSA\text{-}RSA$, we use a (simplified) *single signer (and aggregator) instantiation* of $SA\text{-}RSA$ [15] (also see Remark 1 in Section 4). However, for the sake of completeness, we give the full description of $SA\text{-}RSA$ for multiple-signers below.

Definition 5. *Sequential Aggregate RSA (denoted as* $SA\text{-}RSA$*) 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-RSA.Sig*$(sk_u, \{m_i\}_{i=1}^{u-1}, m_u, \{pk_i\}_{i=1}^{u-1}, pk_u, \sigma_{1,u-1})$*: 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 on* $\overrightarrow{m} = (m_1, \ldots, m_u)$ *under* $\overrightarrow{pk}$ *as* $h_u = H(\overrightarrow{m}||\overrightarrow{pk})$ *and* $y_u = h_u + \sigma_{1,u-1}$*. The sequential aggregate signature outputs the signature* $\sigma_{1,u} \leftarrow y_u^{d_u} \bmod n_u$*.*
- $c \leftarrow$ *SA-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 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* $c = 0$*.*

3 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-CMA* experiment, which we call Immutable-*EU-CMA* (*I-EU-CMA*) experiment.

Definition 6. *I-EU-CMA experiment 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* $\overrightarrow{m}_j = (m_{j,1}, \ldots, m_{j,l})$ *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-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, the forgery is valid and m^* *has not been queried 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-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.*

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.

PISB-CSA-RSA Scheme: An effective way to provide the signature immutability is to compute a protection signature over all data items associated with the query. That is, the server computes a signature on all data items in the query with his own private key. He then aggregates the protection signature over the aggregate signature computed from data owner's signatures. Breaking the immutability of this final aggregate signature is as difficult as forging the server's protection signature [3, 7].

Despite its simplicity, this method is *not* applicable to aggregate signatures such as $C\text{-}RSA$, in which only the signatures computed under the same public key can be aggregated (also called as *non-cross signer* aggregate signature). Recall that $C\text{-}RSA$ cannot aggregate signatures belonging to different signers, since an RSA modulus n can not be safely shared among multiple signers (this leads to the factorization of n, exposing the private keys [16]). 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., $SA\text{-}RSA$ [15]) together with $C\text{-}RSA$. They allow distinct signers to sequentially compute an aggregate signature under distinct public keys (Definition 5).

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'.

One may observe that breaking the immutability of $PISB\text{-}CSA\text{-}RSA$ is as difficult as breaking RSA. We give the formal security analysis of $PISB\text{-}CSA\text{-}RSA$ in Section 5 (Theorem 2).

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

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 $\overline{pk} \leftarrow (\overline{n}, \overline{e})$ such that $n < \overline{n}$. 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 non-cross-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 1. In $PISB\text{-}CSA\text{-}RSA$, we use a *simplified* $SA\text{-}RSA$ variant [15] 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.

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. That is, the server can simply sign the aggregate signature with his private key and define the immutable signature as a signature pair.

$PISB\text{-}Generic$ slightly increases the signature size, since a secondary signature is transmitted along with the aggregate signature. However, this is actually *more communication efficient* than GQ-based and SKROOT-based methods in [3, 7]. That is, a secondary standard signature (e.g., ECDSA [10] with 40 bytes) is much smaller than cryptographic values transmitted (e.g., up to 9 KB) to achieve the immutability in [3,7].

$PISB\text{-}Generic$ also allows the server to choose *any* signature scheme to provide the immutability. For instance, the server may use ECDSA tokens [12] or offline/online signatures [17], which enable very fast response times in demand peaks via pre-computability. This flexibility makes $PISB\text{-}Generic$ more efficient at the server side than existing alternatives (see Table 1). However, note that, iBGLS has slightly smaller signature size (i.e., 20 byte) than that of $PISB\text{-}Generic$ (with the expense of a much higher server computational overhead).

The $PISB\text{-}Generic$ algorithms are defined 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 $\overrightarrow{M} = \{\overrightarrow{m}_1, \ldots, \overrightarrow{m}_u\}$ 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 cross-signer query $\overrightarrow{m} = \{m_1, \ldots, m_k\}$ on a subset of k data owners $U \in \mathbb{U}$. Fetch the corresponding public key and signatures on $\overrightarrow{m}$ from $\overrightarrow{V}$ as $V \leftarrow \{pk_i, m_{i,j}, s_{i,j}\}_{i \in U, \exists j: m_{i,j} \in \overrightarrow{M}}$ 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 $pk \leftarrow \{pk_i\}_{i \in U}$, if $Sig.Ver(\overline{pk}, s', \gamma) = 1$ and $ASig.Ver(\overrightarrow{pk}, \overrightarrow{m}, \sigma) = 1$ hold return $c = 1$, else $c = 0$.

5 Security Analysis

We prove that $PISB$ schemes are $I\text{-}EU\text{-}CMA$ secure in Theorem 1 and Theorem 2. 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 6, $\mathcal{B}$ 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 these 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 $\mathcal{B}$ also loses in the $EU\text{-}CMA$ experiments for $ASig$ and Sig, and therefore $\mathcal{B}$ *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-CSA-RSA* is (t, q_s, ϵ)-*I-EU-CMA* secure if *RSA* signature scheme is $(t', (2 \cdot l)q_s, \epsilon)$-*EU-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-CSA-RSA* query, respectively.

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

We set two separate *EU-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 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-EU-CMA* experiment for *PISB-CSA-RSA* as follows:

- *Setup:* Given $(pk, \overline{pk})$, $\mathcal{B}$ sets the *PISB-CSA-RSA* public key $PK \leftarrow (pk, \overline{pk})$ as in *PISB-CSA-RSA.Kg* algorithm. By Definition 6, $\mathcal{B}$ gives PK to $\mathcal{A}$ and allows $\mathcal{A}$ to ask q_s *PISB-CSA-RSA* signatures under PK.
- Queries: $\mathcal{A}$ queries $\mathcal{B}$ on messages $\overrightarrow{m}_j = (m_1, \ldots, m_l)$ of her choice for $j = 1, \ldots, q_s$. $\mathcal{B}$ handles these queries as follows:
  (a) Given $\mathcal{A}$'s j-th query $\overrightarrow{m}_j$, $\mathcal{B}$ queries $\mathcal{O}_1$ on each m_i and obtains corresponding s_i under pk for $i = 1, \ldots, l$ (as in *PISB-CSA-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-CSA-RSA.Sign*, executed by the server).
- Forgery of $\mathcal{A}$: $\mathcal{A}$ outputs a forgery (m^*, γ^*) and wins the *I-EU-CMA* experiment if
  (i) *PISB-CSA-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-EU-CMA* experiment then $\mathcal{B}$ also loses in the *EU-CMA* experiments for *RSA* against $\mathcal{O}_1$ and $\mathcal{O}_2$, and therefore $\mathcal{B}$ *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-CSA-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) and therefore are not repeated here. □

6 Performance Analysis and Comparison

In this section, we present the performance analysis of $PISB$ schemes and compare them with the existing alternatives. Table 1 (see Section 1) and Table 2 summarize our performance analysis and comparison.

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. Its server side overhead is also plausible and smaller than SKROOT-based scheme. The end-to-end delay of $PISB\text{-}CSA\text{-}RSA$ ($l = 10$) is *50, 65, 41, and 55 times lower* than that of $PISB\text{-}Generic$, GQ-based, SKROOT-based and iBGLS schemes, respectively.

The $PISB\text{-}Generic$ can be instantiated with various signature schemes, which allows different performance trade-offs. BGLS and ECDSA combination achieves both small signature size and high server efficiency, which is a desirable configuration for many applications. Using ECDSA with pre-computation offers *the smallest server response time among all of its counterparts*. However, BGLS increases the signature verification cost, incurring high overhead to the resource-constrained verifiers. Another alternative is to combine $C\text{-}RSA$ and ECDSA, which achieves *both very low server and client computational overheads with the cost of a slightly larger signature size*. ECDSA can be replaced with an online/offline [17] or one-time signature [18], which offers even faster server response with the expense of a very large signature size.

In both $PISB$ and Mykletun et. al. schemes [3, 7], the initialization phases are performed *offline (before the deployment)* by the data owners, whose costs are similar for all schemes and therefore are omitted in this comparison. We focus on the client/server overheads, since they are the online (real-time) overheads and the most important performance metrics for our envisioned applications.

Communication and Storage Overhead: $PISB$ schemes do not require any 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

Table 2. The client and server overhead of $PISB$ and its counterparts for l data items (analytical)

-	Client Comp.	Server Comp.	Sig.	SK	PK
$PISB\text{-}Generic$	$Sig.Ver + ASig.Ver^l$	$Sig.Sign + ASig.Agg^l$	$\|\sigma\|$	$\|sk\|$	$\|pk\|$
$PISB\text{-}CSA\text{-}RSA$	$3Mul + l(H + Mul)$	$Exp_{\|n\|}^{\|n\|} + l \cdot Mul$	$\|n\|$	$2\|n\|$	$2\|n\|$
GQ-based	$3Exp_{\|n\|}^{\|b\|} + l(H + Mul)$	$3Exp_{\|n\|}^{\|b\|} + l \cdot Mul$	$9\|n\|$	$\|b\| + \|n\|$	$\|b\| + \|n\|$
SKROOT-based	$4Exp_{\|2n\|}^{\|n\|} + l(H + Mul)$	$4Exp_{\|2n\|}^{\|n\|} + l \cdot Mul$	$4\|n\|$	$5\|n\|$	$\|n\|$
iBGLS	$(l + 1)(BM + H)$	$EMul + l \cdot EAdd$	$\|q'\|$	$\|q'\|$	$\|p'\| + \|q'\|$

• *Notation*: $Exp_{|y|}^{|x|}$ 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. $ASig.Ver^l$ 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 $2 \cdot EMul$). $|\sigma|$, $|sk|$ and $|pk|$ denote the bit lengths of signature, private key and public key for Sig, respectively (Sig is selected as ECDSA with $|q'|$ in Table 1).
• *Parameters*: Given $\kappa = 80$, we select $|n| = 1024$, $|H| = 160$, $|q'| = 160$, $|p'| = 512$, $|b| = 30$.
• *Multi-round schemes*: 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^*.
• *Measurements*: Table 1 (see Section 1.3) shows the estimated execution times 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 [11] library.

end-to-end delay and client computation overhead). $PISB\text{-}CSA\text{-}RSA$ 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 requirement (introduces a substantial communication delay as shown in Table 1, Section 1). Note that multi-round communication is undesirable for wireless and low bandwidth applications due to the packet loss potential.

Overall, our analysis indicates that $PISB$ schemes are much more efficient and practical than Mykletun et. al. immutable signatures for outsourced database systems.

7 Related Work

Our schemes rely on aggregate signatures as the building block. 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 [19], ID-based for low storage overhead [20]). We discussed aggregate, sequential aggregate [15, 21] and condensed signatures [3, 7] in Section 2.

Mykletun et. al. proposed the first immutable aggregate signatures [3, 7], which have been extensively compared with our schemes in Section 6. Immutable signatures serve as a cryptographic tool for various data outsourcing applications such as database-as-a-service [22] and data protection methods (e.g., [23, 24]). They are also used with other cryptographic primitives such as forward-secure signatures to obtain forward-secure and aggregate logging systems (e.g., [25–27]). Immutability techniques used in these schemes require linear overhead and therefore are not suitable for our envisioned applications.

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., [28, 29]), which are complementary to our work.

8 Conclusion

In this paper, we developed new cryptographic schemes called $PISB$, which provide practical immutable signatures for outsourced databases. $PISB\text{-}CSA\text{-}RSA$ provides the signature immutability with a very low verifier computational overhead, which is ideal for battery/computation limited queriers (e.g., mobile devices). It also offers a very low end-to-end delay, which is desirable for task-intensive applications by increasing the service quality. $PISB\text{-}Generic$ offers various options such as signature pre-computability with its generic construction, which enables a quick server response during query peak times. Both $PISB$ schemes are non-interactive and have small signature sizes. We demonstrate that $PISB$ schemes are much more efficient than previous immutable signatures. Hence, $PISB$ schemes are ideal choices for providing immutability, authentication and integrity services for outsourced database systems.

References

1. Hacigumus, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proceedings of the 18th International Conference on Data Engineering, ICDE 2002, Washington, DC, USA, pp. 29–38 (2002)
2. Sion, R.: Secure data outsourcing. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB), pp. 1431–1432 (2007)
3. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. Transaction on Storage (TOS) 2(2), 107–138 (2006)
4. Patel, A.A., Jaya Nirmala, S., Mary Saira Bhanu, S.: Security and availability of data in the cloud. In: Meghanathan, N., Nagamalai, D., Chaki, N. (eds.) Advances in Computing & Inform. Technology. AISC, vol. 176, pp. 255–261. Springer, Heidelberg (2012)
5. Wang, H., Lakshmanan, L.V.S.: Efficient secure query evaluation over encrypted xml databases. In: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB 2006, pp. 127–138 (2006)
6. Goodrich, M.T., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Privacy-preserving group data access via stateless oblivious ram simulation. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 157–167 (2012)
7. Mykletun, E., Narasimha, M., Tsudik, G.: Signature bouquets: Immutability for aggregated/condensed signatures. In: Samarati, P., Ryan, P.Y.A., Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 160–176. Springer, Heidelberg (2004)
8. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
9. Guillou, L.C., Quisquater, J.-J.: A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 216–231. Springer, Heidelberg (1990)
10. American Bankers Association: ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm, ECDSA (1999)
11. Shamus: Multiprecision integer and rational arithmetic c/c++ library (MIRACL), `http://www.shamus.ie/`

12. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. be improved? Complexity trade-offs with the digital signature standard. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 77–85. Springer, Heidelberg (1995)
13. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman & Hall/CRC (2007)
14. Bellare, M., Rogaway, P.: The exact security of digital signatures: How to sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
15. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004)
16. Ding, X., Tsudik, G.: Simple identity-based cryptography with mediated rsa. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 193–210. Springer, Heidelberg (2003)
17. Catalano, D., Di Raimondo, M., Fiore, D., Gennaro, R.: Off-line/on-line signatures: Theoretical aspects and experimental results. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 101–120. Springer, Heidelberg (2008)
18. Reyzin, L., Reyzin, N.: Better than BiBa: Short one-time signatures with fast signing and verifying. In: Batten, L.M., Seberry, J. (eds.) ACISP 2002. LNCS, vol. 2384, pp. 144–153. Springer, Heidelberg (2002)
19. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006)
20. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.: 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 2007), pp. 276–285. ACM (2007)
21. Zhu, H., Zhou, J.: Finding compact reliable broadcast in unknown fixed-identity networks (short paper). In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 72–81. Springer, Heidelberg (2006)
22. Mykletun, E., Tsudik, G.: Aggregation queries in the database-as-a-service model. In: Damiani, E., Liu, P. (eds.) Data and Applications Security 2006. LNCS, vol. 4127, pp. 89–103. Springer, Heidelberg (2006)
23. Samarati, P., di Vimercati, S.D.C.: Data protection in outsourcing scenarios: issues and directions. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, pp. 1–14 (2010)
24. Thompson, B., Haber, S., Horne, W.G., Sander, T., Yao, D.: Privacy-preserving computation and verification of aggregate queries on outsourced databases. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 185–201. Springer, Heidelberg (2009)
25. Ma, D., Tsudik, G.: A new approach to secure logging. ACM Transaction on Storage (TOS) 5(1), 1–21 (2009)
26. Yavuz, A.A., Ning, P., Reiter, M.K.: BAF and FI-BAF: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems. ACM Transaction on Information System Security 15(2) (2012)
27. Yavuz, A.A., Ning, P., Reiter, M.K.: Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 148–163. Springer, Heidelberg (2012)
28. Ostrovsky, R., Skeith III, W.E.: A survey of single-database private information retrieval: techniques and applications. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 393–411. Springer, Heidelberg (2007)
29. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: Proc. of the 23rd ACM Symposium on Operating Systems Principles, SOSP 2011, New York, NY, USA, pp. 85–100 (2011)