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 [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-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-RP, (iii) We discuss pros and cons of various PISB instantiations by highlighting their performance characteristics.

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., [9]), 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 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 $\text{ASig}$ be a multiplicative aggregate signature scheme (e.g., $C\cdot\text{RSA}$ as in Definition 3). In the above ODB setting, assume that the data owner provides an aggregate signature $\sigma'$ on items $m_1, \ldots, m_k$ to the server (e.g., a music album comprised of different songs). Later, the data owner issues another signature $\sigma''$ 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 $\sigma = \sigma' \cdot \sigma''$. 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 $\sigma$. The term “immutable signature bouquets” $\sigma'$ [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 [11], 1D-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.
We give the details of aggregate, sequential aggregate \cite{13, 16} and condensed signatures \cite{3, 7} in Section 2.

Mykletun et al. \cite{2} 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) \cite{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 signature of knowledge method, which substantially increases the computational cost and has large signature size. Their BGLS signature method iBGLS \cite{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 3.

Immutable signatures serve as a building block for various data outsourcing applications such as database-as-a-service \cite{12} and data protection methods (e.g., \cite{18, 19}). They are also used with other cryptographic primitives such as forward-secure signatures to obtain secure audit logging systems (e.g., \cite{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., \cite{23}) and some special pre-computation techniques (e.g., \cite{23}) are also related to our constructions. In offline/online signatures (e.g., \cite{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., \cite{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 HORST \cite{27} and similar overheads for its variants such as \cite{28-29}). 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 \cite{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 \cite{32} (e.g., for Schnorr signatures \cite{33} but also applicable to several other Meta-ElGamal signatures \cite{34}). Pre-computation methods for RSA-type signatures also have been proposed in \cite{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) \cite{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 \cite{38}. RSA-accumulator \cite{39} based ADSs have been proposed in \cite{40, 41}. Merkle-hash tree \cite{42} is used to construct several ADSs. Verifiable B-trees (i.e., a B-tree using Merkle-hash tree) in \cite{43} and MB-trees (a VB-tree with light hash function instead of heavy signatures) in \cite{44} are some notable Merkle-hash tree based ADS variants. Another line of ADSs is based on authenticated skip-lists \cite{45}, from which several constructions have been developed (e.g., \cite{46, 47}). Kupcu et. al. in \cite{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 \cite{48}.

ADSs generally use a proper digital signature scheme as a building block, where the prover returns a proof (i.e., a signature) per-item in the answer. Remark that, as mentioned in \cite{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-CSA-RSA) can serve as a compact and non-malleable building block for RSA-accumulator based ADSs (e.g., \cite{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., \cite{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-CSA-RSA, (ii) a generic scheme called PISB_Generic, and (iii) a scheme that enables efficient immutable aggregate signature pre-computation called PISB-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-CSA-RSA incurs only 1KB communication overhead, while GQ-based scheme in \cite{3, 4} 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.
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 [23] 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 $k$-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 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. $x \leftarrow S$ denotes that variable $x$ is uniformly selected from set $S$. $|S|$ denotes the cardinality of set $S$. $(x_1, \ldots, x_i)$ denotes $(x_0, \ldots, x_i)$. We denote by $\{0, 1\}^*$ the set of binary strings of any finite length.

Definition 1 A signature scheme $\text{Sig}$ is a tuple of three algorithms $(\text{Kg}, \text{Sign}, \text{Ver})$ defined as follows:
- $(sk, pk) \leftarrow \text{Kg}(1^\kappa)$: Given the security parameter $1^\kappa$, the key generation algorithm returns a private/public key pair $(sk, pk)$ as the output.
- $s \leftarrow \text{Sign}(sk, m)$: The signing algorithm takes $sk$ and a message $m$ as the input. It returns a signature $s$ as the output.
- $c \leftarrow \text{Ver}(pk, m, s)$: The 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 $\text{Sig}$ is defined as follows:
- Setup. Challenger algorithm $B$ runs the key generation algorithm as $(sk, pk) \leftarrow \text{Kg}(1^\kappa)$ and provides $pk$ to the adversary $A$.
- Queries. Beginning from $j = 1$ and proceeding adaptively, $A$ queries $B$ on any message $m_k$ of her choice up to $q_{sk}$ messages. For each query $j$, $B$ computes $s_j \leftarrow \text{Sign}(sk, m_j)$ as the signing oracle of $A$ and returns $s_j$ to $A$.
- Forgery. Finally, $A$ outputs a forgery $(m^*, s^*)$ and wins the EU-CMA experiment, if $\text{Sig}.\text{Ver}(pk, m^*, s^*) = 1$ and $m^*$ was not queried to $B$.

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

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

The EU-CMA experiment for $\text{ASig}$ is a straightforward extension of Definition [2] in which $A$ is required to produce a
forgery under a public key \( pk \in \mathbb{P} \) 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 (\( K_g, \text{Init}, \text{Sign}, \text{Ver} \)) defined as follows:

- \((SK,PK) \leftarrow PISB.Kg(1^n)\): Given the security parameter \( 1^n \), the key generation algorithm runs \((sk,PK) \leftarrow \text{ASig.Kg}(1^n)\) and \((sk,pk) \leftarrow \text{Sig.Kg}(1^n)\). It returns a private/public key pair \( SK = (sk,sk) \) and \( PK = (pk,pk) \), respectively, as the output.

- \( V \leftarrow \text{PISB.Init}(\tilde{M},sk,PK)\): The initialization algorithm takes a message \( \tilde{M} = (m_1,\ldots,m_n) \) and a set of signers \( s_0 \leftarrow \text{ASig.Sign}(sk_i,m_i), i = 1,\ldots,u \) for \( k = 1 \) to \( n \). It returns the set of signers \( \tilde{V} = (\tilde{M},\tilde{S} = (s_i)_{i=1}^{n},PK) \), as the output.

- \( \gamma \leftarrow \text{PISB.Sig}(sk,\tilde{m},\tilde{V})\): The signing algorithm takes \( sk \), messages \( \tilde{m} \in \tilde{M} \) and \( \tilde{V} \) as the input. It computes individual signatures \( \sigma_i = \text{Sig.Sign}(sk_i,m_i), i = 1,\ldots,u \) and \( m,\sigma \leftarrow PISB.\text{Agg}(sk,\tilde{m}) \). It returns the signature \( \gamma = (\sigma) \) as the output.

- \( \gamma \leftarrow \text{PISB.Sig}(sk,\tilde{m},\tilde{V})\): The verification algorithm takes \( PK \), messages \( \tilde{m} \in \tilde{M} \) and \( \tilde{V} \) as the input. If \( \sigma \) is valid under \( \tilde{m} \), it outputs a bit \( c = 1 \) meaning valid; otherwise \( c = 0 \) meaning invalid.

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

**Definition 5** C-RSA is a tuple of three algorithms (\( K_g, \text{Sign}, \text{Ver} \)) defined as follows:

- \((sk,pk) \leftarrow \text{C-RSA.Kg}(1^n)\): Given the security parameter \( 1^n \), 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 \( e,d \in \mathbb{Z}_n \) satisfies \( e \cdot d \equiv 1 \mod \phi(n) \), where \( \phi(n) = (p-1)(q-1) \). The key generation algorithm returns \( sk = (n,d) \) and \( pk = (n,e) \), as the output.

- \( \sigma \leftarrow \text{C-RSA.Sign}(sk,\tilde{m})\): Given \( sk \) and messages \( \tilde{m} = (m_1,\ldots,m_l) \), the signing algorithm returns a signature \( \sigma \in \prod_{j=1}^{l} s_j \mod n \) as the output, where \( s_j \leftarrow H(m_j)^d \mod n \) for \( j = 1,\ldots,l \). \( H \) is a full domain hash function (e.g., [52]) defined as \( H : \{0,1\}^* \rightarrow \mathbb{Z}_n \).

- \( \gamma \leftarrow \text{C-RSA.Ver}(pk,\tilde{m},\sigma)\): Given \( pk = (n,e) \), \( \tilde{m} \) and \( \sigma \), if \( \sigma^e = \prod_{j=1}^{l} H(m_j) \mod n \) then the signature verification algorithm outputs \( \gamma = 1 \) else \( \gamma = 0 \).

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

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

- \((sk,pk) \leftarrow \text{SA-RSA.Kg}(1^n)\): Given the security parameter \( 1^n \) and a set of signers \( U = \{1,\ldots,u\} \), 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 \( 2^{k-1}(1+i/\epsilon) \leq n_i < 2^{k-1}(1+i/\epsilon) \), for \( i = 1,\ldots,u \). It returns a private/public key pair \( sk \leftarrow \{n_i,d_i\}_{i=1}^u \) and \( pk \leftarrow \{n_i,e_i\}_{i=1}^u \) as the output.

- \( \sigma_{1,n} \leftarrow \text{SA-RSA.ASig}(sk_i,m_i,pk_i)\): The signer \( u \) receives aggregate signature \( \sigma_{1,u} \) on messages \( \{m_i\}_{i=1}^u \) under public keys \( \{pk_i\}_{i=1}^u \). The signer \( u \) first verifies \( \sigma_{1,n} \) with the verification algorithm \( \text{SA-RSA.Ver} \). If it succeeds, the signer \( u \) computes the signature as \( h_u = H(\tilde{m}||pk) \) and \( y_u = h_u + \sigma_{1,n} \). The sequential aggregate signature algorithm outputs \( \sigma_{1,u} = y_u \mod n_u \).

- \( c \leftarrow \text{SA-RSA.Ver}(\tilde{m},pk,\sigma_{1,u})\): Given \( \sigma_{1,u} = y_u \mod n_u \) under public keys \( pk = \{n_i,e_i\}_{i=1}^u \), first check \( 0 \leq \sigma_{1,u} \leq n_i \). If \( gcd(\sigma_{1,n},n_u) = 1 \) then \( y_u = \sigma_{1,n} \mod n_u \) else \( y_u = \sigma_{1,n} \).

Compute \( h_u = H(\tilde{m}||pk) \) and \( \sigma_{1,n} = (y_u - h_u) \mod n_u \). Verify signatures recursively as described in this verification algorithm until the base case \( u = 1 \), in which check \( (\sigma_{1,1} - h_1) \mod n_1 = 0 \) where \( h_1 \leftarrow (m_1||pk_1) \). If it holds return \( c = 1 \) else \( c = 0 \).

In our PISB-CSA-RSA scheme, we use a (simplified) single signer (and aggregator) instantiation of SA-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 immutability 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-CMA experiment, which we call Immutability-EU-CMA (I-EU-CMA) experiment.

**Definition 7** I-EU-CMA for PISB is defined as follows:

- **Setup.** Challenger algorithm \( B \) runs \((SK,PK) \leftarrow PISB.Kg(1^n)\) and provides \( PK \) to the adversary \( A \).

- **Queries.** \( A \) queries \( B \) on any message \( \tilde{m}_j = (m_1,\ldots,m_l) \) of her choice for \( j = 1,\ldots,q_s \). \( B \) replies each query \( j \) with a signature \( \gamma_j \) computed under \( PK \).

- **Forgery.** \( A \) outputs a forgery \( (m^*,\gamma^*) \) and wins the EU-CMA experiment, if

(i) \( PISB.\text{Ver}(PK,\gamma^*) = 1 \),

(ii) \( m^* \not\equiv \{\tilde{m}_j\}_{j=1}^{q_s} \mod n \) or \( \exists j \subseteq \{1,\ldots,q_s\} : m^* \not\equiv \{\tilde{m}_j\}_{j=1}^{q_s} \mod n \).

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 \( \{\tilde{m}_1,\ldots,\tilde{m}_{q_s}\} \).

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

**3.2 Aggregate Signature Extraction and Signature Immutability**

A 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.
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 $\sigma'$ from a given of $\sigma_{1,u}$.

**Definition 8** AE experiment for $A\text{Sig}$ is defined as follows:

- **Setup.** Challenger algorithm $B$ runs $(sk, pk) \leftarrow A\text{Sig}.Kg(1^* )$ and provides $pk$ to algorithm $A$.

- **Queries.** $A$ queries $B$ on any batch message $m_j = (m_{1,j}, \ldots, m_{u,j})$ of her choice for $j = 1, \ldots, q_B$. $B$ replies each query $j$ with a signature $\sigma_j$ computed under $pk$.

- **Aggregate Extraction.** $A$ outputs a message-signature pair $(m^*, \sigma')$, where $m^* = (m^*_1, \ldots, m^*_k)$, $1 \leq k \leq u$ and wins the AE experiment, if
  1. $A\text{Sig}.\text{Ver}(\{pk_i, m^*_i\} \in \{1, \ldots, k\}, \sigma') = 1$,
  2. $\forall i \in \{1, \ldots, q_B\} : m^* \subseteq \{m_j \mid j \in I\}$,
  3. $\forall i \in \{1, \ldots, q_B\} : [m^*[[[m_j]_i]]] \neq \{m_j^*\}_j=1^k$.

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.

$A\text{Sig}$ is $(t, q_B, \epsilon)$-AE secure, if no $A$ in time $t$ making at most $q_B$ signature queries has an advantage at least with probability $\epsilon$ 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 DLB-based schemes [53]. 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-General 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-General** 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-General 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-General more efficient at the server side than previous alternatives (see Table 1).

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

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

2) $V \leftarrow \text{PISB-General.Init}(\vec{m}, sk, PK)$: Let messages $\vec{m} = \{m_1^*, \ldots, m_u^*\}$ be database elements to be outsourced, where each $m_i^* = (m_{1,i}, \ldots, m_{k,i})$ belongs to the data owner $1 \leq i \leq u$. Each data owner $i$ computes $s_{i,j} \leftarrow A\text{Sig}.\text{Sign}(sk, m_{i,j})$ for $i = 1, \ldots, u$ and $j = 1, \ldots, t$. Let $V \leftarrow (M, S, PK)$ and provide $V$ to the server, where $S = \{s_{i,j}\}_{i=1}^u \times \{j=1, \ldots, t\}$.

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

4) $c \leftarrow \text{PISB-General.Ver}(pk, \vec{m}, \gamma)$: Given $\gamma = (\sigma, s')$ and $pk \leftarrow \{pk_i\}_{i \in U}$, if $\text{Sig}.\text{Ver}(pk, s', \gamma) = 1$ and $A\text{Sig}.\text{Ver}(pk, \vec{m}, \sigma) = 1$ 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-General, the server may compute the protection signature as $s' \leftarrow \text{Sig}.\text{Sign}(sk, \sigma||ts_p)$, where $ts_p$ 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 $s_{i,j} \leftarrow A\text{Sig}.\text{Sign}(sk_i, m_{i,j}||i||j)$ 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-General 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
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-RSA is an efficient scheme, its immutable variants (e.g., [3,2]) are inefficient as discussed in Section 4.2.

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-RSA [16]), as defined in Definition 3 together with C-RSA. We call our new scheme that exploits this observation as PISB-CSA-RSA.

In PISB-CSA-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-RSA signature $\sigma'$ by aggregating RSA signatures. The server then uses SA-RSA to compute an immutable aggregate signature $\gamma$ on $m_1, \ldots, m_l$ with his keys $(\pi, d)$ by aggregating it on $\sigma'$. The public key of the system is $\{(n, e), (\pi, \tau)\}$. The verification order of the client is with SA-RSA under $(\pi, \tau)$ for $\gamma$ and then with C-RSA under $(n, e)$ for $\sigma'$.

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

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

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

2) $\overrightarrow{V} \leftarrow$ PISB-CSA-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 $\overrightarrow{V} = (\overrightarrow{m}, \overrightarrow{S})$ and provide $\overrightarrow{V}$ to the server, where $\overrightarrow{S} = (s_1, \ldots, s_l)$.

3) $\gamma \leftarrow$ PISB-CSA-RSA.$Sign(sk, \overrightarrow{V})$: 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 $h \leftarrow H(\overrightarrow{m})[\overrightarrow{pk}], y \leftarrow (h + \sigma') \bmod \pi$ and $\gamma \leftarrow y^2 \bmod \pi$.

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

Remark 2 In PISB-CSA-RSA, we use a simplified SA-RSA variant [16] with the following properties: (i) SA-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, $n_1$ 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., [23] or offline/online signatures (e.g., [23]) to PISB setting. However, as discussed in Section 4.3, special pre-computation methods are not applicable to various PISB instantiations such as C-RSA and pairing-based constructions; and offline/online signatures introduce extremely large public key and signature sizes. Hence, none of these methods is suitable to accelerate signature generation in PISB schemes.

To address these limitations, we developed Random Pre-computed PISB (PISB-RP) scheme. PISB-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 $\sigma$ 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 $\sigma$, which is more efficient than computing $s$ itself.

Notice that PISB-RP does not rely on costly one-time signatures (e.g., 3-5 KB signature and public key overhead for each message), and therefore is much more communication and storage efficient than PISB-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-Generic. However, unlike PISB-Generic (but like offline/online and other non-generic pre-computed tokens [23]), it requires storing a pair as $(r, s)$ for each message to be signed at the server side.

The PISB-RP algorithms are defined below.

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

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

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

- **Offline Phase**: The server pre-computes $(r, s)$ to be used in online phase, where $r \overset{\$}{\leftarrow} \{0, 1\}^n$, $s \leftarrow$ Asig.$Sign(\overrightarrow{sk}, r)$ 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 $\subseteq U$. Fetch the corresponding public key and signatures on $\overrightarrow{m}$ from $\overrightarrow{V}$ as $V \leftarrow \{pk_i, m_{i,j}, s_{i,j}\}_{i \in U, j \in \overrightarrow{m}}$ 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.$Agg(\overrightarrow{pk}_i, \overrightarrow{m}_i, (\sigma', s))$. 

U.S. Government work not protected by U.S. copyright.
4) $c \leftarrow \text{PISB-RP.}(PK, \bar{m}, \gamma)$: Given $\gamma = (\sigma, r)$ and $pk \leftarrow \{(pk_j)_{j \in U}, pk\} \subseteq PK$, if $|r| = \kappa$ and $\text{ASig.}(pk, \bar{m}, r), \sigma = 1$ hold return $c = 1$, else $c = 0$.

We now discuss two specific instantiations of PISB-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-RP).

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

Notice that, compared to iBGLS [7, 12] and PISB-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-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-RSA signature of the data owner): At the initialization phase PISB-RP.Init, the data owner generates $r \leftarrow \{0, 1\}^\gamma, s \leftarrow [H(r)]^a \mod n$ and gives $(r, s)$ to the server (this corresponds to the offline phase of PISB-RP.Sign executed by the server). At the signature generation (online) phase of PISB-RP.Sign, given aggregate signature $\sigma'$ on $\bar{m}$, the server computes $\sigma \leftarrow \sigma' \cdot s \mod n$ and set $\gamma = (\sigma, r)$.

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 multiplications plus hash operations.

5 SECURITY ANALYSIS

We prove that PISB schemes are I-EU-CMA secure in Theorem 1 and Theorem 2 in ROM [50]. We ignore terms that are negligible in terms of $\kappa$.

**Theorem 1** PISB-Generic is $(t, q_s, \epsilon)$-I-EU-CMA secure, if ASig is $(t', q_s, \epsilon)$-EU-CMA secure and Sig is $(t', q_s, \epsilon)$-EU-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 $A$ breaks $(t, q_s, \epsilon)$-I-EU-CMA secure PISB-Generic. We then construct a simulator $B$, which breaks $(t', q_s, \epsilon)$-EU-CMA secure ASig or $(t', q_s, \epsilon)$-EU-CMA secure Sig by using $A$ as subroutine.

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

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

- **Setup**: Given $(pk, \bar{pk})$, $B$ sets the PISB-Generic public key $PK \leftarrow \{(pk_j)_{j \in U}, \bar{pk}\}$ as in PISB-Generic. $Kg$ algorithm. By Definition 7 $B$ gives $PK$ to $A$ and also permits $A$ to make $q_s$ PISB-Generic signature queries.

- **Queries**: $A$ queries $B$ on messages $m_j = (m_{j,1}, \ldots, m_{j,9})$ of her choice for $j = 1, \ldots, q_s$. $B$ handles queries as follows:
  a) Given $A$’s $j$-th query $m_j$, $B$ queries ASig.Sign oracle on $\bar{m}_j$ under $\bar{pk}$. The ASig.Sign oracle returns $s_j, i \leftarrow \text{ASig.}Sig.(sk_i, m_{j,i})$ for $i = 1, \ldots, u, B$ then computes the aggregate signature $\sigma_j \leftarrow \text{ASig.}Agg(pk, \bar{m}_j, s_{j,1}, \ldots, s_{j,u})$. This step is identical to PISB-Generic.Init algorithm, where $M$ in this experiment is comprised of $u$ vectors each with $q_s$ data items.
  b) $B$ queries Sig.Sign oracle on $\sigma_j$ under $\bar{pk}$. The Sig.Sign oracle returns $\bar{s}_j \leftarrow \text{Sig.}Sign.(\bar{sk}_j, \sigma_j)$ (as in PISB-Generic-Sign algorithm, executed by the server). $B$ replies $A$ with $\gamma_j = (\sigma_j, r_j)$.

- **Forgery of $A$**: $A$ outputs a forgery $(m^*, \beta^*) = (\sigma^*, s'^*)$ and wins the $I$-EU-CMA experiment if
  i) PISB-Generic. Ver$(PK, m^*, \beta^*) = 1$,
  ii) $m^* \not\subseteq \{\bar{m}_1, \ldots, \bar{m}_{q_s}\}$ or $\forall I \subseteq \{1, \ldots, q_s\} : m^* \subseteq \{\bar{m}_k\}_{k \in I}$.

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

a) If $m^* \not\subseteq \{\bar{m}_1, \ldots, \bar{m}_{q_s}\}$ then $B$ returns the forgery $(m^*, \sigma^*)$ against ASig, which is non-trivial since $B$ did not ask $m^*$ to ASig.Sign. This forgery is valid since PISB-Generic. Ver$(PK, m^*, \beta^*) = 1$ implies ASig. Ver$(pk, m^*, \sigma^*) = 1$.

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

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

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

**Theorem 2** PISB-CSA-RSA is $(t, q_s, \epsilon)$-I-EU-CMA secure if RSA signature scheme is $(t', (2l)q_s, \epsilon)$-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 $A$ breaks $(t, q_s, \epsilon)$-I-EU-CMA secure PISB-CSA-RSA. We construct a simulator $B$ breaking $(t', (2l)q_s, \epsilon)$-EU-CMA RSA by using $A$ as subroutine.

We set two separate $EU$-CMA experiments for $B$, in which it is given RSA public keys $pk = (n, e)$ and $\bar{pk} = (\bar{n}, \tau)$ and provided signature oracles under their corresponding private
Theorem 3 \(\text{PISB-RP is } (t, q_s, e)\)-I-EU-CMA secure, if \(\text{ASig is } (t', q_s, e)\)-EU-CMA secure and \((t, q_s, e)\)-AE secure, where \(t' = O(t) + q_s \cdot \text{Op and Op denotes the cost of signature generation for } \text{ASig.}\)

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

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

- Setup: Given \((pk, \bar{pk}),\) \(\mathcal{B}\) sets the \(\text{PISB-CSA-RSA}\) public key \(PK \leftrightarrow (pk, \bar{pk})\) as in \(\text{PISB-CSA-RSA.Kg}\) algorithm. By Definition 7 \(\mathcal{B}\) gives \(PK\) to \(\mathcal{A}\) and allows \(\mathcal{A}\) to ask \(q_s \text{ PISB-RP signatures under } PK.\)

- Queries: \(\mathcal{A}\) queries \(B\) on messages \(m_j = (m_j, \ldots, m_{j, u})\) of her choice for \(j = 1, \ldots, q_s.\) \(\mathcal{B}\) handles queries as follows:
  a) \(\mathcal{A}\) generates \(r_j \leftarrow \{0, 1\}^n\) and sets \(m_j \leftrightarrow (m_j, r_j).\) \(\mathcal{B}\) inserts \(r_j\) and \(m_j\) into \(\mathcal{L}_R\) and \(\mathcal{L}_D,\) respectively.
  b) \(\mathcal{A}\) queries \(\text{ASig.Sign} \text{on } m_j\) and obtains its corresponding signature \(\sigma_j.\) \(\mathcal{B}\) sets \(\gamma_j \leftarrow (\sigma_j, r_j)\) and returns \(\gamma_j\) to \(\mathcal{A}.\)

- Forgery of \(\mathcal{A} : \mathcal{A}\) outputs a forgery \((m^*, \gamma^*)\) and wins the \(\text{I-EU-CMA experiment if}\)
  i) \(\text{PISB-RP.Ver}(PK, m^*, \gamma^*) = 1,\)
  ii) \(m^* \notin \{m_1, \ldots, m_{q_s}\}\) or \(\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq \|k \in 1, m_k \}

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

a) If \(m^* \notin \{m_1, \ldots, m_{q_s}\}\) then \(\mathcal{B}\) returns the forgery \((m^*, s^*)\) against \(\mathcal{O}_1,\) where \(s^*\) is computed from \(\sigma^*\) 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 \((y^*, \gamma^*)\) 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 \(\text{PISB-CSA-RSA.Ver}(PK, m^*, \gamma^*) = 1 \implies m^* \text{ and } \gamma^* \text{ are valid under } PK \text{ and } \bar{PK},\) respectively.

b) If \(\exists I \subseteq \{1, \ldots, q_s\} : m^* \subseteq \|k \in 1, m_k \|\) then \(\mathcal{B}\) returns the forgery \((\sigma^*, \gamma^*)\) against \(\mathcal{O}_2.\) This forgery is valid and non-trivial as discussed above.

The execution time and probability analysis are similar to Theorem 1 (i.e., the simulation is perfectly indistinguishable).
that of PISB-Generic, GQ-based, SKROOT-based and iBGLS schemes, respectively. The end-to-end delay of PISB-RP instantiated with $C$-RSA is $(1 + 1)H + (2l + 4) \cdot \text{Md}$. This is the lowest computational end-to-end delay among all compared alternatives being a magnitude(s) of times more efficient than previous schemes \cite{7}, \cite{13} 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-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-Generic, and only the aggregate signature itself in PISB-CSA-RSA. The private and public key sizes are the sum of that of their base signature schemes.

PISB-Generic with BGLS and ECDSA has the smallest key and signature sizes among its counterparts with the client computation as seen in Table 2. PISB-CSA-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-RP requires transmitting $\kappa$-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., \cite{23}, \cite{25}, \cite{27}). These signature size and pre-storage overhead are more efficient than that of PISB-Generic instantiation with ECDSA token and offline/online signatures (e.g., \cite{23}, \cite{25}, \cite{27}) as shown in Table 1 and Table 2. However, notice that, all pre-computation methods including PISB-RP increases the server storage compared to PISB schemes and previous schemes (e.g., \cite{7}, \cite{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-Generic instantiated with $C$-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 \cite{27} with some variants such as TV-HORS \cite{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 \cite{29}. In PISB-RP instantiated with $C$-RSA, the size of private/public key are $nl$ but not $2nl$, 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-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-CSA-RSA is a versatile scheme with high computational efficiency and plausible signature/key sizes without requiring pre-computation. We select PISB-RP instantiated with $C$-RSA, since it achieves the highest computational efficiency, low communication and end-to-end delay overall, but it requires pre-storage. Notice that PISB-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-Generic is instantiated with BGLS \cite{9} as $ASig$ (20 byte) and with ECDSA \cite{55} as $Sig$ (40 byte protection signature) for pre-computed parameters (0.36 ms) \cite{59} or pre-computed tokens \cite{24} (0.03 ms). PISB-RP in this example is instantiated with $C$-RSA. PISB-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 $|n| = 1024$, $|H| = 160$, $|g'| = 160$, $|p'| = 512$, $|b| = 30$, $z = 20$, $t = 256$. 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 \cite{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 $Z^n_\kappa$. 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-RP, offline/online signatures (e.g., \cite{23}, \cite{25}, \cite{26}) and PISB-Generic instantiations that rely offline/online signatures and ECDSA tokens \cite{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-RP with $C$-RSA, PISB-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
### 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


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


13


U.S. Government work not protected by U.S. copyright.
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.