Implementation of Verified Set Operation Protocols Based on Bilinear Accumulators

Luca Ferretti(✉), Michele Colajanni, and Mirco Marchetti

Department of Engineering "Enzo Ferrari",
University of Modena and Reggio Emilia, Modena, Italy
{luca.ferretti,michele.colajanni,mirco.marchetti}@unimore.it

Abstract. This paper proposes an efficient protocol for verifiable delegation of computation over outsourced set collections. It improves state of the art protocols by using asymmetric bilinear pairing settings for improved performance with respect to previous proposals based on symmetric settings. Moreover, it extends update operations by supporting efficient modifications over multiple sets. With respect to previous work the proposed protocol has a modular design, that clearly identifies its main building blocks and well-defined interfaces among them. This novel conceptualization allows easier auditing of the protocol security properties and serves as the blueprint of a novel implementation that is released publicly (https://weblab.ing.unimore.it/people/ferretti/versop/). To the best of our knowledge, this is the first public implementation of a protocol for verifiable sets operations.

1 Introduction

Many approaches for securing distributed systems focus on controlling network and system activities [2,3,9,15], and do not rely on cryptography. Moreover, most applications of cryptography to data outsourcing scenarios focus on confidentiality [11,12]. On the other hand, the proposed protocol guarantees the correctness of results in scenarios where data and computation are delegated to an untrusted server. With respect to previous protocols proposed in literature [7,19], this paper proposes three main contributions.

This is the first protocol for verifiable set operations that relies on asymmetric bilinear pairings, while all previous proposals leverage symmetric bilinear parings. Asymmetric settings are preferable, since they are characterized by lower computational costs, thus resulting in performance optimization for the whole protocol. Moreover, this is the first protocol for verifiable sets operations that provides efficient support for insertions, deletions and updates over multiple sets at once. This is achieved by designing a variant update protocol for accumulation trees that allows the owner to provide an aggregate proof for multiple update operations. Finally, while previous works describe a *monolithic* protocol, with no high-level components and interfaces among them, in this paper we model the proposed protocol as a combination of three modular components, each exposing well-defined interfaces.

S. Foresti and G. Persiano (Eds.): CANS 2016, LNCS 10052, pp. 626–636, 2016.
DOI: 10.1007/978-3-319-48965-0_41

We release a public implementation of the protocol described in this paper based on open-source cryptographic libraries [1,16,22], that we extend and wrap to obtain modular components with higher-level interfaces. To the best of our knowledge, this is the first public implementation of a protocol for verifiable operations over a collection of sets.

The remainder of this paper proceeds as follows. Section 2 provides a short description of the main cryptographic primitives that form the basis for our protocol. Section 3 describes the reference scenario, introduces its main actors and gives an overview of the proposed protocol and of its main modules. Sections 4 and 5 describe the details of the protocol modules. Finally, Sect. 6 outlines concluding remarks and propose future work.

2 Cryptographic Building Blocks

Polynomial Representations of Sets. A set X can be represent through a characteristic polynomial $C_X(s) = \prod_{x \in X}(x + s)$, where s is a formal variable that is used as secret information in cryptographic protocols, and the elements of the set $x \in X$ are the addition opposite of the polynomial solutions [13,18]. The polynomial is defined over Z_p, where p is a large prime number. We denote as $h_z(\cdot)$ and $\phi_z(\cdot)$ hash functions that accept as inputs arbitrary binary strings and elliptic curve elements, and that produce elements in $Z_p - \{0\}$. We assume that those functions are automatically applied when the elements of the input set X of $C_X(s)$ are not in Z_p. The polynomial $C_X(s)$ can be also represented and computed through its coefficient form. By denoting the coefficients as $\{a_i\}_{i=[|X|]}$, the characteristic polynomial of set X is computed as $C_X(s) = \sum_{i=1}^{|X|} a_i \cdot s^i$. Coefficients can be computed efficiently from its roots by using FFT interpolation algorithms [21]. Our implementation wraps algorithms of the NTL library [22] and integrates them with hash functions of the Charm framework [1] to provide high-level interfaces to compute characteristic polynomials of sets defined over the most common data domains.

Bilinear Pairings. In this paper we focus on asymmetric pairing settings (either Type 2 or Type 3 pairings [8,14]) that are usually faster than symmetric pairings adopted by previous work [7]. We denote as $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \hat{e})$ the public parameters that define an asymmetric bilinear pairing setting. Let g_1, g_2 be generators of cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order p (that we represent as multiplicative), $\mathbb{G}_T$ a multiplicative cyclic group of the same order and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be the pairing function that satisfies the following properties: bilinearity: $\hat{e}(m^a, n^b) = \hat{e}(m, n)^{ab} \forall m, n \in \mathbb{G}_1 \times \mathbb{G}_2, \forall a, b \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$; non-degeneracy: $\hat{e}(g_1, g_2) \neq 1$; computability: there exists an efficient one-way algorithm to compute $\hat{e}(m, n)$, $\forall m, n \in \mathbb{G}_1 \times \mathbb{G}_2$. Our implementation is based on the Charm cryptographic framework [1], that wraps the *PBC* library [16]. We plan to extend our implementation with faster open source backend libraries, such as [5,17].

Bilinear Accumulators. Informally, a cryptographic accumulator is a small constant size data structure (a *digest*) that can authenticate an arbitrary number of values [4]. In this paper we are interested in bilinear (map) accumulators (BMA) [18] (also *set accumulators*), that implement valid cryptographic accumulators based on bilinear pairings and characteristic polynomial representations of sets. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \hat{e})$ be the parameters of the asymmetric bilinear pairing setting, $s \in \mathbb{Z}_p^*$ be the secret key and $[g_1^s, \ldots, g_1^{s^q}, g_2^s, \ldots, g_2^{s^q}]$ be the public key, where q is the maximum number of values that can be stored in the accumulator. The BMA of a set X, that we denote as f_X, can be computed by using two algorithms: $f_{sk}(X)$, that uses the secret key s, and $f_{pk}(X)$, that only uses the public key, as following: $f_{sk}(X) = g_1^{C_X(s)} = g_1^{\prod_{i=1}^{|X|}(x_i+s)}$, $f_{pk}(X) = \prod_{i=1}^{|X|}\left(g_1^{s^i}\right)^{a_i}$. To prove that a value $x \in X$ is stored in the BMA, a party (that knows the whole set X) must produce a *witness* $w_Y \in \mathbb{G}_2$ such that $\hat{e}(f_{pk}(x), w_Y) \stackrel{?}{=} \hat{e}(f_X, g_2)$, where w_Y is the BMA of the set $Y = X \backslash \{x\}$ computed over $\mathbb{G}_2$. The equation to compute w_Y, that we denote as $w_{pk}(Y)$, is: $w_{pk}(Y) = \prod_{i=1}^{|Y|}\left(g_2^{s^i}\right)^{a_i}$, where $\{a_i\}_{i \in [|Y|]}$ is the set of the coefficients of the characteristic polynomial $C_Y(s)$. Both $f_{pk}(\cdot)$ and $w_{pk}(\cdot)$ are BMA functions that are usually represented by the same notation in symmetric pairing settings. We denote as f_X and w_X elements of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Our implementation of BMAs protocols extends those of characteristic polynomials and bilinear pairings.

Extractable Collision-Resistant Hash Functions. An *extractable collision resistant hash* (ECRH) function is a cryptographic function that can produce a *succinct non-interactive argument-of-knowledge* (SNARK) to demonstrate the correctness of some simple computation [6]. In this paper we are interested in ECRH functions that prove the correct computation of BMAs. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \hat{e})$ be the parameters of the asymmetric bilinear pairing setting, $(s, \alpha) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ be the secret key and $[g_1^s, \ldots, g_1^{s^q}, g_2^s, \ldots, g_2^{s^q}, g_1^\alpha, g_1^{\alpha s}, \ldots, g_1^{\alpha s^q}, g_2^\alpha, g_2^{\alpha s}, \ldots, g_2^{\alpha s^q}]$ be the public key. We denote as F_X the ECRH of set X. It is computed as $F_X = (f_X, f'_X)$ [6], where f_X is the BMA of set X and f'_X is the BMA of set X computed with public key $[g_1^{\alpha s}, \ldots, g_1^{\alpha s^q}]$, as $f'_{pk}(X) = \prod_{i=1}^{|X|}\left(g_1^{\alpha s^i}\right)^{a_i}$. As discussed in [6], function f'_X represents a proof of correct computation for the BMA f_X based on security assumptions that extend the *knowledge of exponent* assumption [10], first described to guarantee chosen-ciphertext security of asymmetric encryption. We denote as F_X, f'_X the black-box output of the functions $(f_{pk}(X), f'_{pk}(X))$ and $f'_{pk}(X)$. ECRH functions can be verified publicly through a pairing operation: $\hat{e}(f_X, g_2^\alpha) \stackrel{?}{=} \hat{e}(f'_X, g_2)$. Note that the public key of our protocol does not uses the array of elements $[g_2^\alpha, g_2^{\alpha s}, \ldots, g_2^{\alpha s^q}]$. From an implementation perspective, ECRH functions are bilinear accumulators. Thus, our implementation adds an additional higher-level interface to that of BMAs.

3 Scenario and Protocol Overview

We assume that an organization that owns data (*owner*) outsources data and computations to an external *server*. Outsourced data are in the form of a collection of sets. Each set is associated to a label and can include one or more elements. Outsourced data can be queried by one or more *users*, that interact directly with the *server*. A query is an arbitrary combination of set operations expressed over the input sets. Any composition of set operations can be modeled by an abstract syntax tree (AST), where leaves are input sets and intermediate nodes are set operations. The AST corresponding to an example query "$(A \cup B \cup C) \cup (D \cap E)$" issued by the *user* is shown in Fig. 1. The leaves represent the sets involved in the query, referred by their labels, while intermediate nodes are the three set operations. Each output edge of an intermediate node represents the intermediate result of the corresponding set operation and the input of another set operation. The output of the root node represents the plaintext data returned to the *user*. The *server* proves the correctness of the inputs (π'_{leaf}) and of the operations ($\pi'_{\cap}, \pi'_{\cup}$) through different specialized routines that use bilinear accumulators to represent input and output sets. The proposed protocol builds a chain of such proofs along the vertexes of the AST, thus proving correctness of the whole computation. We distinguish three main categories of proofs.

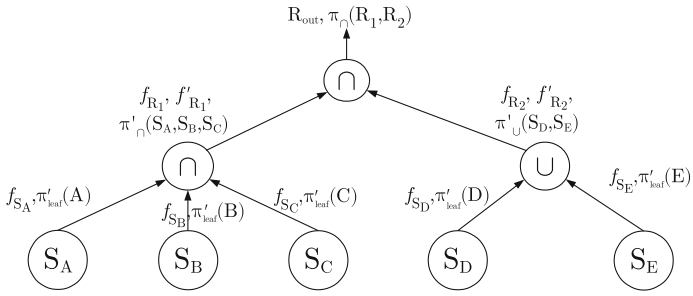

Fig. 1. AST and proofs for a verified hierarchical set operation

Proofs of correctness for the input sets. A *user* does not know any content stored in the sets collection, except the set of the available labels used to issue queries. Given a query by a *user*, the *server* returns BMAs for all input sets involved in the query and proofs of correctness that demonstrate that each BMA represents the set associated to the requested label. The proposed protocol produces these proofs by using an *accumulation tree*, that we describe in Sect. 4.

Proofs of correct computation for single set operations. Given a single set operation, the *server* is able to produce proofs of correct computation based on the BMAs of the input sets. A *user* can verify the correctness of the output by knowing the authenticated BMAs that represent the input sets and the proof for the set operation. We describe single set operation protocols in Sect. 5.

Proofs of knowledge for all intermediate results. To bind the output of a set operation as the input of another set operation, the proposed protocol produces proofs of knowledge for all intermediate results. Sections 4 and 5, describe how the *server* produces proof of correctness for intermediate results.

4 Accumulation Tree Protocols

An accumulation trees is an authenticated data structure based on constant size N-ary trees that allow efficient authentication of data by building many levels of hierarchical authentication structures, each authenticating the lower one by using cryptographic accumulators as intermediate nodes [20]. The proposed protocol leverages accumulation tree based on bilinear accumulators to authenticate all the input sets involved in a query issued by a *user*. Accumulation trees support three operations: *setup*, *update* and *leaf queries*. Setup and update operations are used to initialize and modify the accumulation tree accordingly with the content of the sets collection. Leaf queries allow to prove correctness of the inputs used in queries issued by *users*. **Notation.** We refer to an accumulation tree as A_k, where k is the *version* of the tree (the number of update operations). The tree has $m = |D|$ leaves, each representing a set of the sets collection and identified by a label $\ell \in \mathcal{L}$. Figure 2 shows an example accumulation tree based on an N-ary tree that authenticates a sets collection of $m = 27$ sets. We identify a node as $v[i, j]$, where i is its level ($i = 0$ is the level of the root, $i = 1$ of the root's children, ...) and j is its position within the level. We define t as the lowest level of the tree. The following functions identify relevant sets of nodes: $N(v[i, j])$ and $P(v[i, j])$ return the children and the parent of $v[i, j]$; $R(v[i, j])$ returns the nodes in the path from $v[i, j]$ to the root; $J(i)$ returns the number of nodes at the level i. **Setup.** The *owner* computes each leaf of the accumulation tree $v[t, \ell]$ as the BMA that contains the elements of the set S_ℓ and a unique representative of the label ℓ:

$$v[t, \ell] = f_{sk}(S_\ell \cup \{\ell\}) = g_1^{(h_z(\ell)+s)\prod_{x \in S_\ell}(h_z(x)+s)}, \ \forall \ell \in L \tag{1}$$

The *owner* then computes each non-leaf node as the BMA of its children $N(v[i, j])$:

$$v[i, j] = f_{sk}(N(v[i, j])) = g_1^{\prod_{x \in N(v[i,j])}(\phi_z(x)+s)}, \forall i = t-1, \ldots, 0, \forall j = 1, \ldots, J(i) \tag{2}$$

The *owner* sends the accumulation tree to the *server* as the authentication structure A_0, and its root $v[0, 0]$ to the *users* as the digest d_0. The *owner* maintains the accumulation tree locally to execute updates on the *server*.

Update. The implemented update protocol improves the one described and used in [7,19,20] by allowing insertion and deletion of multiple elements on many sets through a single operation, and producing a single proof demonstrating the correctness of all updates at once. Since the updated version of the accumulation

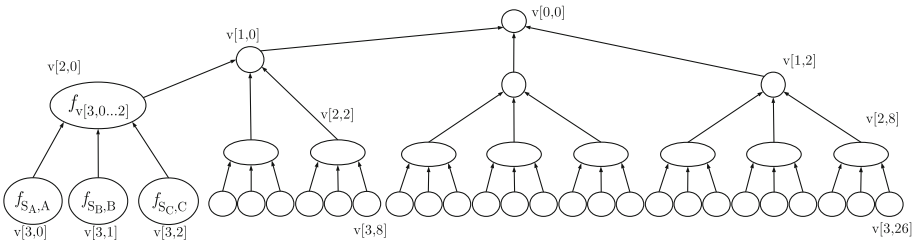

Fig. 2. Example of accumulation tree for $m = 27$ sets and degree equal to 3 ($\varepsilon = 1/3$)

tree generated by the proposed protocol is equal to that the original protocol, the security proofs proposed for the original protocol still hold.

We model the update operation $\mathcal{U}$ as an associative array $\{\ell : (add_\ell, del_\ell)\}$, where add_ℓ is the set of values inserted in S_ℓ and del_ℓ is the set of values deleted from S_ℓ. To update the leaves of the accumulation tree, the *source* computes the characteristic polynomials of inserted ($C_{add_\ell}(s)$) and deleted values ($C_{del_\ell}(s)$) for each set. These polynomials are used to update each leaf $v[t, \ell]$ as following:

$$v'[t, \ell] = v[t, \ell]^{C_{add_\ell}(s) \cdot C_{del_\ell}(s)^{-1}}, \ \forall \ell \in \mathcal{U} \tag{3}$$

Then the *owner* updates the intermediate nodes of the accumulation tree in the path from an updated leaf to the root. All the updated leaves and intermediate nodes are stored in the *upd* data structure. The *owner* maintains locally the new version of the accumulation tree A_{k+1} and sends only *upd* to the *server*. After the *server* confirms the update, the *owner* can delete the old version A_k and distribute its root as the new digest d_{k+1} to the *users*.

Leaf query. We distinguish two variants of leaf queries: those used to guarantee correctness of plaintext sets returned to *users*, and those used to guarantee correctness of input sets used in hierarchical queries. We denote the routines that implement the protocols for plaintext results as *queryTreePlaintext* and *verifyTreePlaintext*, and those for intermediate results as *queryTreeNode* and *verifyTreeNode*. Given a label ℓ, the *server* uses the *queryTreePlaintext* (*queryTreeNode*) protocol to return the set S_ℓ (the BMA f_{S_ℓ}) and the proof π_{leaf} ($\pi\prime_{leaf}$) that authenticates the set (the BMA) with respect to the accumulation tree A_k. A *user* that knows the digest d_k can execute a protocol *verifyTreePlaintext* (*verifyTreeNode*) to verify the correctness of S_ℓ (f_{S_ℓ}).

The *server* builds π_{leaf} by including all nodes in the path from the requested leaf to the root, and witnesses that authenticate the chain of nodes. We denote as v_t the leaf corresponding to ℓ ($v[t, \ell]$), and $v_{t-1}, \ldots, v_1$ the nodes in the path from the leaf to the root (excluded). We denote as γ_i the witness that binds v_i to v_{i+1}, computed as the BMA of the children of node v_i except the node whose correctness we must prove, that is v_{i-1}:

$$\gamma_i = f_{pk}(N(v_i) \backslash \{v_{i-1}\}) \tag{4}$$

To prove the chain of nodes $v_t, \ldots, v_1$, the *server* computes witnesses $\gamma_{t-1}, \ldots, \gamma_0$. The complete proof π_{leaf} is:

$$\pi_{leaf} \equiv \pi'_{leaf} = ((v_t, \gamma_{t-1}), (v_{t-1}, \gamma_{t-2}), \ldots, (v_1, \gamma_0)) \tag{5}$$

The routine for intermediate results *queryTreeNode* includes all the described operations, but it also requires the *server* to compute the BMA of the set f_{S_ℓ} as $f_{pk}(S_\ell)$. This value differs from v_t because it does not include the representative of the label ℓ (see Eq. (1)).

The verification routines involve three phases: *(a)* results verification (plaintext or their BMA) by using the leaf node v_t; *(b)* verification of $v_t, \ldots, v_1$ by using the witnesses $\gamma_{t-1}, \ldots, \gamma_1$; *(c)* verification of v_1 by using the witness γ_0 and the digest d_k (that is v_0) trustfully obtained by the *owner*.

$$\begin{aligned}
&(a) \qquad \hat{e}(f_{pk}(S_\ell), g_2^{h_z(\ell)} \cdot g_2^s) \stackrel{?}{=} \hat{e}(v_t, g_2) \\
&(b) \qquad \hat{e}(\gamma_i, g_2^{\phi_z(v_{i+1})} \cdot g_2^s) \stackrel{?}{=} \hat{e}(v_i, g_2), \forall i = t-1, \ldots, 1 \\
&(c) \qquad \hat{e}(\gamma_0, g_2^{\phi_z(v_1)} \cdot g_2^s) \stackrel{?}{=} \hat{e}(d_k, g_2)
\end{aligned} \tag{6}$$

If any of the previous conditions is not verified, the *user* rejects the results returned by the *server*.

5 Verified Set Operations

The proposed protocol supports *union* and *intersection* set operations. To demonstrate the correctness of these operations through BMAs, it is necessary to express them in terms of operations among their characteristic polynomials. To solve this issue, we reduce unions and intersections to a combination of primitive operations that we can prove through characteristic polynomials: *subset*, *multiset union* and *disjointness*.

Subset and multiset union. Let us consider sets A and X such that $A \subset X$. By construction, C_X (the characteristic polynomial of X) is divisible by C_A, thus there exists a witness polynomial W such that $C_X = C_A \cdot W$. The proof of a subset relation is the witness W, computed as the characteristic polynomial of the set $B = X \backslash A$. Verification is computed through the bilinear pairing function.

Another operation that we can prove by using a single witness is multiset union. Let us consider two input sets A and B and their multiset union $X = A + B$ (informally referred to as *multiset concatenation* in [7]). The output set X includes duplicate elements if A and B are not disjoint. Multiset union can be mapped to a multiplication operation between the characteristic polynomials of A and B, as $C_X = C_A \cdot C_B$. Given X, (or its BMA f_X), the proof of multiset union only includes the BMAs of the input sets A and B computed in the correct bilinear group ($\mathbb{G}_1$ or $\mathbb{G}_2$). Verification can be computed as $\hat{e}(f_A, w_B) \stackrel{?}{=} \hat{e}(f_X, g_2)$ or $\hat{e}(f_B, w_A) \stackrel{?}{=} \hat{e}(f_X, g_2)$. Subset and multiset union operations only support two input sets due to the nature of the pairing function $\hat{e}$.

Set disjointness. The proof for sets disjointness can be reduced to a proof of divisibility between polynomials: if and only if the intersection between the sets is the empty set, then the *gcd* of the characteristic polynomials of the sets is equal to 1. Consider sets $S_1, \ldots, S_n$. If $\cap_{i\in[n]} S_i = \emptyset$, then the *gcd* of the characteristic polynomials $C_{S_i}, \forall i \in [n]$ is equal to 1. Hence, there exists unique polynomials $\dot{q}_i$ such that $\sum_{i\in[n]}(C_{S_i} \cdot \dot{q}_i) = 1$. Polynomials $\{\dot{q}_i\}$ can be computed by executing iteratively the extended euclidean algorithm for finite field couples of polynomials [19]. Our implementation is based on the extended euclidean algorithms provided by the NTL library [22]. The proof for sets disjointness $\pi_\emptyset(S_1, \ldots, S_n)$ and its verification are as following:

$$\pi_\emptyset(S_1, \ldots, S_n) = (w_{\dot{q}_1}, \ldots, w_{\dot{q}_n}) \tag{7}$$

$$verifyDisjoint(\pi_\emptyset, f_{S_1}, \ldots, f_{S_n}) : \prod_{i\in[n]} \hat{e}(f_{S_i}, w_{\dot{q}_i}) \stackrel{?}{=} \hat{e}(g_1, g_2) \tag{8}$$

Set intersection. We consider input sets $S_1, \ldots, S_n$ and a set I that is the output of set intersection $I = \bigcap_{i\in[n]} S_i$. If I is empty or equal to one of the input sets, the operation can be reduced to a disjointness or a subset proof. Otherwise, the proof relies on two properties: I is a subset of all sets: $I \subset S_i, \forall i \in [n]$; the set complements of each set S_i are disjoint to I: $\bigcap(S_i \backslash I) = \emptyset$. In the following we distinguish proofs computed for plaintext results ($\pi_\cap$), from those computed for intermediate results ($\pi'_\cap$).

Proof $\pi_\cap$ includes the proof of disjointness $\pi_\emptyset$ for all the set complements, plus the BMAs of all the set complements:

$$\pi_\cap = (f_{S_1\backslash I}, \ldots f_{S_n\backslash I}, \pi_\emptyset(S_1 \backslash I, \ldots, S_n \backslash I)) \tag{9}$$

Proof $\pi'_\cap$ includes the proof of disjointness $\pi_\emptyset$ for all the set complements, plus the ECRH functions of all the set complements. Due to the construction of ECRH functions we can denote $\pi'_\cap$ through two equivalent notations:

$$\pi'_\cap(I, S_1, \ldots, S_n) = (w_I, F_{S_1\backslash I}, \ldots F_{S_n\backslash I}, \pi_\emptyset) \equiv (w_I, f'_{S_1\backslash I}, \ldots f'_{S_n\backslash I}, \pi_\cap) \tag{10}$$

Verifying $\pi_\cap$ requires verifying the subset and disjunction properties:

$$\begin{aligned} &verifyIntersectionPlaintext(\pi_\cap, I, f_{S_1}, \ldots, f_{S_n}) : \\ &\quad \forall i \in [n], \hat{e}(f_{S_i\backslash I}, w_{pk}(I)) \stackrel{?}{=} \hat{e}(f_{S_i}, g_2), \quad \prod_{i\in[n]} \hat{e}(f_{S_i}, w_{\dot{q}_i}) \stackrel{?}{=} \hat{e}(g_1, g_2) \end{aligned} \tag{11}$$

Verifying $\pi'_\cap$ requires to verify the ECRH functions, the witness of the intersection and the plaintext intersection proof $\pi_\cap$ (with the small variant of already having the witness $w_I = w_{pk}(I)$ available):

$$\begin{aligned} &verifyIntersectionNode(\pi'_\cap, f_I, f_{S_1}, \ldots, f_{S_n}) : \\ &\quad \hat{e}(f_I, g_2) \stackrel{?}{=} \hat{e}(g_1, w_I), \qquad \forall i \in [n], \hat{e}(f_{S_i\backslash I}, g_2^\alpha) \stackrel{?}{=} \hat{e}(f'_{S_i\backslash I}, g_2), \\ &\quad verifyIntersectionPlaintext(\pi_\cap, w_I, f_{S_1}, \ldots, f_{S_n}) \end{aligned} \tag{12}$$

Set union. We consider two input sets A and B and the set $U = A \cup B$. It is possible to prove set union by using the set inclusion-exclusion principle [7]: $A \cup B = (A + B) \backslash (A \cap B)$. Since $(A \cap B) \subseteq (A + B)$ by construction, the set difference operation can also be proved as the multiset union $A + B = (A \cup B) + (A \cap B)$. Thus, set union proof must include an intermediate intersection proof $(A \cap B)$, knowledge proofs for the intermediate result $A \cap B$) and witnesses for multiset union.

$$\pi'_{\cup}(A, B) = (\pi'_{\cap}, F_{A\cap B}, w_{A\cap B}, w_B), \qquad \pi_{\cup}(A, B) = (\pi'_{\cap}, F_{A\cap B}, w_B) \tag{13}$$

Plaintext and intermediate proofs are verified in similar ways, where verification of intermediate results require a pairing operation to test the correctness of element $w_{A\cap B}$:

$$\begin{aligned} &verifyUnionPlaintext(\pi_{\cup}, U, f_A, f_B) : \\ &\quad \hat{e}(f_{A\cap B}, g_2^{\alpha}) \stackrel{?}{=} \hat{e}(f'_{A\cap B}, g_2), \qquad \hat{e}(f_B, g_2) \stackrel{?}{=} \hat{e}(g_1, w_B), \\ &\quad \hat{e}(f_A, w_B) \stackrel{?}{=} \hat{e}(f_{A\cap B}, w_{pk}(U)), \\ &\quad verifyIntersectionNode(\pi'_{\cap}, f_{A\cap B}, f_A, f_B) \end{aligned} \tag{14}$$

$$\begin{aligned} &verifyUnionNode(\pi'_{\cup}, f_U, f_A, f_B) : \\ &\quad \hat{e}(f_{A\cap B}, g_2^{\alpha}) \stackrel{?}{=} \hat{e}(f'_{A\cap B}, g_2), \qquad \hat{e}(f_{A\cap B}, g_2) \stackrel{?}{=} \hat{e}(g_1, w_{A\cap B}), \\ &\quad \hat{e}(f_B, g_2) \stackrel{?}{=} \hat{e}(g_1, w_B), \qquad \hat{e}(f_A, w_B) \stackrel{?}{=} \hat{e}(f_U, w_{A\cap B}), \\ &\quad verifyIntersectionNode(\pi'_{\cap}, f_{A\cap B}, f_A, f_B) \end{aligned} \tag{15}$$

Note that set union natively supports only two inputs: in the case of union operations among multiple sets, the query must be handled as a hierarchical query composed by multiple binary operations.

6 Conclusions

This paper describes the implementation of a protocol for efficient verifiable delegation of set operations based on bilinear accumulators. We extended literature by detailing a modular implementation that identifies the main building blocks of the protocol and defines standard interfaces. We extend the original protocols by proposing a variant for asymmetric bilinear pairings and an improved update protocol for multiple sets. We implemented the protocol and released it publicly. This is the first public implementation of a protocol for verifiable sets operations.

Acknowledgments. This work was supported by MAECI-CyberLab-2015/2016.

References

1. Akinyele, J.A., Garman, C., Miers, I., Pagano, M.W., Rushanan, M., Green, M., Rubin, A.D.: Charm: a framework for rapidly prototyping cryptosystems. J. Crypt. Eng. **3**(2), 111–128 (2016). http://charm-crypto.com/

2. Andreolini, M., Colajanni, M., Marchetti, M.: A collaborative framework for intrusion detection in mobile networks. Inf. Sci. **321**(C), 179–192 (2015)
3. Andreolini, M., Colajanni, M., Pietri, M., Tosi, S.: Adaptive, scalable and reliable monitoring of big data on clouds. J. Parallel Distrib. Comput. **79**(C), 67–79 May 2015
4. Benaloh, J., De Mare, M.: One-way accumulators: a decentralized alternative to digital signatures. In: Proceedings of IACR CRYPTO (1993)
5. Beuchat, J.-L., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In: International Conference on Pairing-Based Cryptography, 20 July 2016. https://github.com/herumi/ate-pairing
6. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Proceedings of 2012 ACM Third International Conference on Innovations in Theoretical Computer Science (2012)
7. Canetti, R., Paneth, O., Papadopoulos, D., Triandopoulos, N.: Verifiable set operations over outsourced databases. In: Proceedings of 2014 IACR International Conference on Public-Key Cryptography (2014)
8. Chatterjee, S., Hankerson, D., Menezes, A.: On the efficiency and security of pairing-based protocols in the type 1 and type 4 settings. In: Hasan, M.A., Helleseth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 114–134. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13797-6_9
9. Colajanni, M., Gozzi, D., Marchetti, M.: Enhancing interoperability and stateful analysis of cooperative network intrusion detection systems. In: Proceedings of ACM Symposium on Architecture for Networking and Communications (2007)
10. Damgård, I.B.: Towards practical public key systems secure against chosen ciphertext attacks. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 445–456. Springer, Heidelberg (1992). doi:10.1007/3-540-46766-1_36
11. Ferretti, L., Colajanni, M., Marchetti, M.: Distributed, concurrent, and independent access to encrypted cloud databases. IEEE Trans. Parallel Distrib. Syst. **25**(2), 437–446 (2014)
12. Ferretti, L., Pierazzi, F., Colajanni, M., Marchetti, M.: Scalable architecture for multi-user encrypted sql operations on cloud database services. IEEE Trans. Cloud Comput. **2**(4), 448–458 (2014)
13. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Proceedings of IACR CRYPTO (2004)
14. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Appl. Math. **156**(16), 3113–3121 (2008)
15. Lodi, G., Querzoni, L., Baldoni, R., Marchetti, M., Colajanni, M., Bortnikov, V., Chockler, G., Dekel, E., Laventman, G., Roytman, A.: Defending financial infrastructures through early warning systems: the intelligence cloud approach. In: Proceedings of 5th ACM Workshop CSIIRW (2009)
16. Lynn, B.: On the implementation of pairing-based cryptosystems. Ph.D. thesis, Stanford University, 20 July 2016. https://crypto.stanford.edu/pbc/
17. Naehrig, M., Niederhagen, R., Schwabe, P.: New software speed records for cryptographic pairings. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 109–123. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14712-8_7
18. Nguyen, L.: Accumulators from bilinear pairings and applications. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 275–292. Springer, Heidelberg (2005). doi:10.1007/978-3-540-30574-3_19

19. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal verification of operations on dynamic sets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 91–110. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22792-9_6
20. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Authenticated hash tables. In: Proceedings of 15th ACM Conference on Computer and Communications Security (2008)
21. Preparata, F.P., Sarwate, D.V.: Computational complexity of fourier transforms over finite fields. Math. Comput. **31**(139), 740–751 (1977)
22. Shoup, V.: NTL: a library for doing number theory, 20 July 2016. http://www.shoup.net/ntl/