An Efficient and Scalable Authenticated Cloud Storage Scheme Based on Trapdoor Hash Functions

Santosh Chandrasekhar and Mukesh Singhal

University of California, Merced

{*schandrasekhar@ucmerced.edu, msinghal@ucmerced.edu*}

Abstract

Storage services are among the primary cloud computing offerings, providing advantages of scale, cost and availability to its customers. However, studies and past experiences show that large-scale storage service can be unreliable, and vulnerable to various threats that cause loss and/or corruption of customer data. Since cloud service providers have an incentive to hide these corruptions/losses to protect their reputation, explicit mechanisms are necessary to verify the integrity of the stored data. In this work, we present an authenticated cloud storage system that is designed for services where the data is populated by multiple sources and retrieved by the clients. The goal is to allow the clients to verify the authenticity and integrity of the retrieved data in a scalable and efficient way, without requiring implicit trust on the storage service provider. The proposed mechanism, based on our recently proposed multi-trapdoor hash functions, utilizes a third-party trustworthy authentication service provider to generate authentication tags (that are analogous to signatures) for the data stored at the storage service. The communication and computation overheads associated with authenticating the data retrieved by clients remain constant regardless of the data size, or the number of sources. The proposed scheme achieves this by generating tags of individual files and aggregating tags of multiple files (from multiple sources) using a novel mechanism based on multi-trapdoor hashing scheme. We develop a discrete log-based instantiation of the scheme and evaluate its security and performance. Our security analysis shows that forging the individual or aggregate authentication tags is infeasible under the discrete log assumption. Our performance comparison with other schemes in the literature demonstrates that the proposed scheme achieves superior efficiency and scalability, with constant cost associated with computing and verifying the individual and aggregate authentication tags, and constant size of the aggregate authentication tag, regardless of the number of data files or the number of sources.

1 Introduction

Users of today's clouds are diverse, ranging from single users to large organizations, utilizing different types of services at various levels of abstraction. Along with the usual infrastructure, platform, software, and the subsequent derivative "as-a-service" offerings, clouds also provide a rich array of services, tools and middleware to clients for controlling, managing, monitoring and maintaining their assets, which include rented resources, software and other services. This rich array of cloud offerings, along with its benefits of scale, cost and accessibility, are the key contributing factors to the continued growth of cloud adoption and utilization. This growth includes data storage, a service that continues to be among the most popular ones, for purposes of content distribution, data analysis, archiving, among many others. A cloud-based storage system saves customers the overhead associated with the hardware, software, resources and professionals that would be necessary for a traditional in-house data storage solution, and instead, offloads these overheads to the cloud storage service provider (CSSP).

Motivation: Primary research challenges involving cloud-based storage systems concern security and performance (that covers aspects of scalability, availability, latency, and usability). Security plays a key role in adoption of technology, particularly with clouds [7, 16, 35, 46]. Security concerns in clouds stem from clients relinquishing direct control over its assets (information and system components) to the cloud service provider leading to risk of potential mismanagement of those assets, and a shared multi-tenant environment that can cause exposure of client's data and resources to other consumers. Security threats can originate both externally and internally. External security threats result from attacks that target exposed interfaces due to resources, applications and data being hosted on public domains that are accessed/administered over the Internet, and from increase in system complexity leading to a large attack surface. Internal security threats can originate from within a client's organization (current/former employees, affiliates, etc.) utilizing a cloud service, cloud provider's organization, as well as from other clients using the service. These threats can be both malicious and unintentional causing loss and/or corruption of a client's data. Moreover, cloud providers have a strong incentive to hide any loss/corruption to protect their reputation, and maintain its profits [42]. Thus, when a client stores its data with an external, potentially untrusted, CSSP, explicit mechanisms are necessary to protect client assets from external and internal attacks on its integrity, authenticity, availability, authorization and privacy (or confidentiality). While each of these security concerns are important and have been addressed in the past, most scenarios, at the very least, require assurances of data integrity and authenticity [36].

Problem Statement: This paper focusses on the problem of ensuring integrity, authenticity and non-repudiation of data in cloud-based storage systems in an efficient and scalable, which we collectively term as *lightweight*, manner. Primary factors that influence overheads include computation at all entities involved (client, data source and storage system), bandwidth between client and storage system, and space requirements to store authentication information [36]. Efficiency implies that resource consumption (bandwidth, storage, and processing) remain low for each operation, and scalability implies that this consumption does not increase with growth in the size of the system, encompassing items like, amount of data or entities involved.

Existing literature contains several solutions that deal with problems of storage integrity as well as query integrity [47] for databases that are outsourced. We cover existing research work more extensively later in the paper. Unfortunately, we find that a large majority of these solutions only consider storage systems comprising a single source of data and do not scale well when multiple data sources are present. The need to consider multiple data sources cannot be overlooked, since it is relevant in several real world scenarios. A well-known example is a data warehouse that integrates data that is generated through an assortment of activities by several disparate sources. This data is then used by clients to support reporting, analytics, forecasting, and decision making. Another example is the electronic medical (or health) record systems being implemented by various US health care providers as part of national health care reform [1]. These systems comprise of a database of electronic health records of various patients from various healthcare providers. For instance, the APeX (Advanced Patient-Centered Excellence) system used by the University of California San Fransisco (UCSF) Medical Center involves 168 clinics, two hospitals and an Orthopaedic institute. The small number of proposals addressing data integrity in outsourced databases that *do* provide support for multiple sources [36, 37, 38], do not scale well. To fill this gap in existing research, our goal is to develop a lightweight mechanism to ensure the integrity and authenticity of data retrieved by clients over insecure public networks from a cloud storage system that is not necessarily trusted and is storing data from *multiple sources*.

Contributions: We develop a novel mechanism for authentication of outsourced data with support for multiple sources that achieves (near-) constant communication and computation overheads regardless of the data size, or the number of sources. The proposed scheme achieves this by generating authentication tags (that are analogous to signatures) of individual files and aggregating tags of multiple files (from multiple sources) using a novel mechanism based on the recently proposed paradigm called a multi-trapdoor hashing scheme [11].

Unlike traditional hash functions, a trapdoor hash function [30] is associated with a public and private key, also called a hash and trapdoor key, respectively. Using the hash key, any entity can compute a hash of a given message. Without the knowledge of the trapdoor key, a trapdoor hash function is collision resistant. However, given the trapdoor key along with the trapdoor hash value, collisions can be computed efficiently.

Based on the concept of double-trapdoor hash functions, Chandrasekhar et al. [11] proposed the concept of multi-trapdoor hash functions, where the hash function is associated with multiple trapdoors, each belonging to a different entity. Such a trapdoor hash function allows one to compute the hash of a message (or set of messages) using hash keys belonging to those multiple entities (who own the respective trapdoor keys), to generate what we call a target trapdoor hash value. Given this, each entity can now compute a collision with the target hash value using their respective trapdoor keys, and combine their respective collisions to generate a multi-trapdoor hash collision between the original and a new set of messages.

The proposed scheme exploits properties of multi-trapdoor hash function to generate authentication tags for individual files and aggregate them for multiple files. Each authentication tag represents the result of a trapdoor collision between hashes of the file and a common message (for e.g., the sources' identities) shared between all data sources. More precisely, the authentication tag of a file comprises of values that, when used for computing the trapdoor hash of the file, results in a digest that equals the trapdoor hash of the common message. This collision can only be computed by the source holding the trapdoor key, and thus, the tag represents a signature on the file. When a query arrives, we combine the tags corresponding to the files in the query response to generate a multi-trapdoor hash collision between the query response and the common shared message. The size of aggregate tag – values necessary to compute the multi-trapdoor hash of the query response – remains constant regardless of the number of files in the response, or the number of data sources. Verification now involves checking whether the multi-trapdoor hash values of the common message and the query response match. Once again, the cost associated with computing the multi-trapdoor hash value of the query response remains constant. This results in a scheme that is highly scalable in terms of computation and communication overheads regardless of the data size, or the number of sources. In summary, the contributions of this paper are as follows:

1. Using our recently proposed multi-trapdoor hashing scheme [11] we develop a novel mechanism for authenticating query response in cloud-based storage systems, where data is populated by multiple data sources. The scheme allows clients to verify the integrity and authenticity of files returned by an outsourced database in response to their queries.

2. The proposed scheme requires a small fixed sized authentication tag for each file in the storage, and, unlike existing schemes in the literature, incurs (near-) constant computation and communication overheads for verifying the query response regardless of the number of files returned by the query, or the number of sources.

3. We evaluate the security of the proposed scheme and prove that forging the individual or aggregate authentication tags is infeasible under the discrete log assumption. We also demonstrate that the proposed scheme out-performs other existing schemes that provide similar features (in particular, support for multiple data sources) in terms of scalability, which is among the most important properties when considering cloud-based storage systems.

Organization: The rest of the paper is organized as follows. We discuss related work in Section 2. In Section 3 we present background material on trapdoor hash functions [30], Schnorr signatures [40] and a Schnorr-based multisignature scheme [27, 34] that are used for constructing the proposed authenticated cloud storage system. In Section 4, we review a technique to allow multiple nodes to find collisions with the hash of a given message. We then apply this technique in Section 5 to build an efficient and scalable trapdoor hash-based scheme for authenticating query responses in cloud-based storage systems. In Section 6, we conduct an analysis of the proposed authenticated cloud storage scheme including its correctness, security evaluation and the performance of the proposed scheme. Section 7 concludes the paper with pointers to future research.

2 Related Work

We segment the related work into two areas, namely, authentication in outsourced databases and trapdoor hash functions.

Authentication in Outsourced Databases: Ensuring integrity and authenticity of outsourced storage has been a well-studied problem, with a wide variety of solutions that include the use of authenticated skip lists [19, 26, 45], aggregate signatures [36, 37, 38], homomorphic authenticators [2, 28, 41], hash-based approaches [17, 21], tree-based approaches [18, 31], and several other techniques (for an extensive list of references, see [2, 29, 32, 44]. While some of these solutions are aimed towards maintaining integrity of remote archival storage without having to retrieve the data (in other words, providing a guarantee of data possession, or storage auditing) [2, 19, 28, 41, 48], other solutions are designed to provide verification of query results in outsourced databases – which is more aligned with our goal. In both cases, the large majority of proposals assume a single source of data. On the other hand, our goal is to provide lightweight query integrity and authenticity in cloud storage systems storing data from multiple sources. One solution can be to extend the existing ideas to support multiple sources. The most promising candidates for such a solution approach are schemes based on authenticated skip lists and tree-based approaches. The general idea here is to build a separate data structure for each source with each root value signed by the respective source. When a query arrives, the query response will not only contain the necessary data structure elements but also the aggregation of the root signatures, that together can verify the authenticity of the query response. Unfortunately, the only available technique to generate such an aggregate signature is by Boneh et al. [6] that, although being efficient in terms of aggregation cost and signature length, incurs verification overhead that grows linearly with the number of signers.

The proposed work is related to the idea first proposed by Mykletun et al. [36] that uses digital signature aggregation to provide lightweight authentication of query replies in outsourced databases. The paradigm of signature aggregation allows combining multiple signatures into a single unified signature, whose verification simultaneously establishes the validity of all component signatures. Aggregate signatures offer bandwidth and storage savings, and are usually more efficient to verify compared to verifying all component signatures individually. In Mykletun et al.'s scheme, the system model comprises of a database service provider, one or more data sources (or owners), and one or more clients (or queriers). A data source signs each tuple in the database, and stores the signature along with the tuple at the service provider. When a client query arrives, the database service provider aggregates the signatures of all tuples in the query response and sends it to the client. Successful verification of this single aggregate signature convinces the client of the authenticity and integrity of the query response. While Mykletun et al.'s scheme is able to scale when the system contains a single data source, when multiple sources are present, the cost of verifying query response increases linearly with the number of sources [36] due to the use of the pairing-based aggregate signature scheme BAS [6]. The same cost increase is also observed in schemes by Narasimha et al. [37] and Pang et al. [38] which also use BAS for signature aggregation. This linear increase in verification cost can result in significant overhead for clients with limited computation capabilities.

Trapdoor Hash Functions: The concept of a trapdoor hash function was originally derived from the notion of trapdoor commitments proposed by Brassard et al. [8]; Krawczyk et al. [30] used trapdoor hash functions (referred to as chameleon hash) to construct a non-interactive non-transferable signature scheme, called chameleon signatures (closely related to undeniable signatures), under the hash-and-sign paradigm. Chameleon signature allows a signer to undeniably commit to the contents of a signed document, but does not allow the recipient of the signature to disclose the signer's commitment to a third party without the signer's consent. The trapdoor hash function employed by Krawczyk et al. suffers from the key exposure problem that allows anyone with the knowledge of a collision to compute the private trapdoor key. Ateniese et al. [4] partially addressed this problem by introducing an identity-based trapdoor hashing scheme that uses a new key pair for each collision computation. This way, a collision only leads to the to the exposure of a single trapdoor key that was used for computing that particular collision, thus, preventing the exposure from affecting other collisions. Later Chen et al. [12] and Ateniese et al. [5] proposed full constructions of trapdoor hash functions without key exposure, and provided several applications of trapdoor hashing. Following this, several key-exposure free hashing schemes have been proposed, and for an extensive list of references and recent developments, see [14].

Shamir et al. [43] employed trapdoor hash functions to develop a new paradigm, called hash-sign-switch, that could be used to convert any signature scheme into an online/offline signature scheme [20]. In online/offline signature schemes, the signature generation procedure is split into two phases that are performed offline (before the message to be signed is known) and online (after the message is known). By shifting the computational burden to the offline phase, online/offline signatures can achieve very high efficiency for signing messages during the online phase. In Shamir et al.'s scheme, the values computed in the offline phase (a trapdoor hash value and its signature) can only be used for a single message in the on-line phase, as multiple uses of the same offline trapdoor hash value for different online messages leads to the disclosure of the trapdoor key. To fix this issue, Chen et al. [13], and later Harn et al. [25] presented multiple-collision trapdoor hashing schemes where revealing multiple collisions of the same hash value doesn't leak the secret trapdoor key. We note that among the various key-exposure-free schemes by Ateniese et al. [5], the one related to twin Nyberg-Rueppel signatures is also a multiple-collision trapdoor hashing scheme – a property which they later term as strongly unforgeable [3].

Trapdoor hashing schemes also find applications in the development of several novel signature schemes that include, threshold signatures [15], proxy signatures [10, 33], sanitizable signatures [3], and stream authentication schemes [9].

3 Background

In this section, we provide a brief description of single- and double- trapdoor hash functions, and a discrete log (DL) based signature and multisignature scheme which will be used as building blocks to construct the proposed authenticated cloud storage system.

3.1 Trapdoor Hash Functions

A trapdoor hashing scheme TH consists of the tuple ($ParGen$, $KeyGen$, TH, $TrapColGen$), that are described next. $ParGen$ is an algorithm for parameter generation that takes a security parameter 1^k as input, and outputs system public parameters params. $KeyGen$ is a key generation algorithm that takes params as input, and outputs a trapdoor and hash key pair (TK, HK). TH is a trapdoor hash function that takes params, HK, a message m, a random element r, as inputs, and outputs the digest of m denoted as $TH_{HK}(m, r)$. $TrapColGen$ is a collision finding algorithm that takes params, TK, message m, random element r, and an additional message $m' \neq m$ as inputs, and outputs a collision parameter r' such that $TH_{HK}(m, r) = TH_{HK}(m', r')$. Figure 1 shows the computation of a trapdoor hash of a message m to get a digest h, along with the computation of a collision parameter r' for a message m', which, when used for computing the trapdoor hash of m', leads to the same digest h.

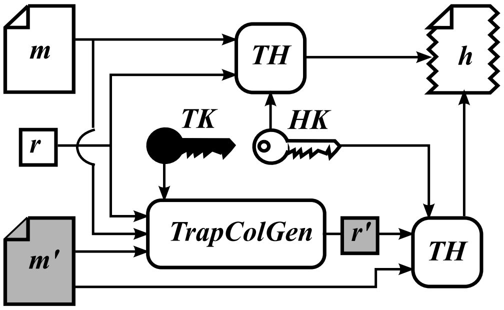


Figure 1: A trapdoor hash function.

For trapdoor hash function to be practical, computing the digest of a message using TH and collisions using $TrapColGen$ must be achievable in polynomial time. The function TH is said to be part of a trapdoor hash family $\mathcal{TH}$ described by params, where each TH is associated to a hash key HK. Well-known security notions associated with trapdoor hashing schemes include collision forgery resistance and key-exposure resistance [5, 30, 43]. Collision forgery resistance [30, 43] implies that given system parameters, params and hash key, HK, it is computationally infeasible to find a tuple $\langle m, m', r, r' \rangle$ such that $TH_{HK}(m, r) = TH_{HK}(m', r')$. Key-exposure resistance implies [5] that given system parameters, params and a tuple $\langle m, m', r, r', HK \rangle$ such that $TH_{HK}(m, r) = TH_{HK}(m', r')$, it is computationally infeasible to find the trapdoor key, TK corresponding to HK.

Krawczyk et al. [30] proposed a simple DL-based trapdoor hashing scheme, DL-TH, which was later used by Shamir et al. [43] in the construction of their online/offline signature scheme. The system public parameters are $\mathsf{params} = \langle p,\, q,\, \alpha,\, H \rangle$, where p and q are primes such that $q \mid p - 1$, α is an element of order q in $\mathbb{Z}_p^*$ and $H : \{0,1\}^* \mapsto \mathbb{Z}_q^*$ is a cryptographic hash function. The (private) trapdoor key and (public) hash key pair of an entity is computed as $(TK, HK) = (x \in_R \mathbb{Z}_q^*, X = \alpha^x \mod p)$. An entity generates a trapdoor hash of a message $m \in \{0,1\}^*$ using the hash key X, by choosing an element $r \in_R \mathbb{Z}_q^*$ and computing the hash as $TH_X(m, r) = \alpha^{H(m)} X^r \mod q$[1]. Given system parameters params, the trapdoor key x, message $m \in \{0,1\}^*$, $r \in \mathbb{Z}_q^*$ and an additional message $m' (\neq m) \in \{0,1\}^*$, an entity finds a collision between m and m' by computing a parameter $r' = x^{-1}(H(m) - H(m')) + r \mod q$. We now have,

$$\begin{aligned}
TH_X(m', r') &= \alpha^{H(m')} X^{r'} \\
&= \alpha^{H(m')} X^{(x^{-1}(H(m) - H(m')) + r)} \\
&= \alpha^{H(m')} \alpha^{H(m) - H(m')} X^r \\
&= \alpha^{H(m)} X^r = TH_X(m, r)
\end{aligned}$$

Given m, r, m' and r', such that $TH_X(m, r) = TH_X(m', r')$, we denote the tuple $\langle m, r, X, m', r',\, X \rangle$ as a trapdoor collision tuple under X, the pair of messages (m, m') as a trapdoor collision producing message pair, and describe the operation as generating a trapdoor collision between m and m', under X.

The DL-TH scheme suffers from the well known *key exposure problem* [5]. Given a trapdoor collision tuple $\langle m, r, X, m', r', X \rangle$, any third party can compute the trapdoor key x corresponding to the hash key X as follows:

$$x = (m - m')(r' - r)^{-1} \mod q$$

Intuitively, this scheme exhibits key exposure because extracting the trapdoor key given a collision involves solving a simple equation with one unknown. To prevent this, we next present the concept of a double-trapdoor hash function that utilizes multiple trapdoors during collision computation, a well-known technique by Ateniese et al. [5].

3.2 Double-Trapdoor Hash Functions

A double-trapdoor hash function, as the name suggests, is associated with a pair of hash keys, one long-term and the other ephemeral (or one-time). Chen et al. [15] were the first to provide a formal definition of a double-trapdoor hash family. Our description of a double trapdoor hashing scheme is a variation on the definition by Chen et al. [15], and extends traditional definitions [43] to allow constructions of trapdoor hash functions that can generate collisions using ephemeral keys. Unlike in Chen et al.'s definition, the proposed definition uses different ephemeral keys when generating collisions (similar to multiple-collision schemes [25]). Thus, our description only permits construction of trapdoor hash functions that do not expose any information that can lead to the computation of additional collisions after revelation of a collision producing message pair [15].

Definition 1 *A double trapdoor hashing scheme,* DTH *has the following components:*

[1] All exponentiations modulo q are shorthand representations for computing exponentiations modulo p followed by reducing the result modulo q. Also, note that the trapdoor hash function can act on arbitrarily long messages as message m can be hashed using a regular collision-resistant hash function before computing its trapdoor hash without affecting any security properties [30].

$ParGen$: *With a security parameter* 1^k *as input, outputs system public parameters* params.

$KeyGen$: *With* params *as input, outputs a long-term trapdoor and hash key pair* (TK_l, HK_l).

$EKeyGen$: *With* params *as input, outputs a one-time (or ephemeral) trapdoor and hash key pair* (TK_e, HK_e).

TH: *With* params, $HK = (HK_l, HK_e)$, *a message* m, *a random element* r, *as inputs, outputs the hash* $TH_{HK}(m, r)$.

$TrapColGen$: *With* params, $TK = (TK_l, TK_e)$, *message* m, *random element* r, *and an additional message* $m' \neq m$ *as inputs, outputs a collision parameter* r' *and* $HK' = (HK_l, HK'_e)$ *such that* $TH_{HK}(m, r) = TH_{HK'}(m', r')$.

Analogous to single-trapdoor hashing schemes, in double-trapdoor hashing schemes, computing the digest of a message using TH and collisions using $TrapColGen$ must be achievable in polynomial time. The function TH is said to be part of a double-trapdoor hash family $\mathcal{TH}$ described by params, where each TH is associated to a hash key $HK = (HK_l, HK_e)$. Similar notions for collision and key-exposure resistance also exist for trapdoor hash functions that use ephemeral (or double) trapdoors [15]. Informally, collision forgery resistance implies that given system parameters, params and hash key, $HK = (HK_l, HK_e)$, it is computationally infeasible to find a trapdoor collision tuple $\langle m, r, HK, m', r', HK' \rangle$, where $HK' = (HK_l, HK'_e)$ and $TH_{HK}(m, r) = TH_{HK'}(m', r')$. Key-exposure resistance implies that given system parameters, params and a trapdoor collision tuple $\langle m, r, HK, m', r', HK' \rangle$ such that $TH_{HK}(m, r) = TH_{HK'}(m', r')$, it is computationally infeasible to find the long-term trapdoor key, TK_l corresponding to HK_l. We re-visit these notions in Section 6.2, where we provide their formal definitions. Figure 2 shows the operation of a double trapdoor trapdoor hash function. The function hashes a message m to get a digest h. During collision computation with a message m', the function outputs a collision parameter r' and an ephemeral hash key HK', which, when used for computing the trapdoor hash of m', leads to the same digest h.

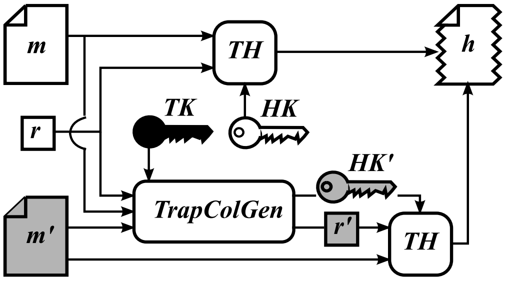


Figure 2: A double-trapdoor hash function. $HK = (HK_l, HK_e)$ and $HK' = (HK_l, HK'_e)$.

We now review the DL-DTH [11] scheme that overcomes the key exposure problem of DL-TH [5]. Similar to DL-TH, the common system public parameters are params $= \langle p, q, \alpha, H, G \rangle$, with the additional cryptographic hash function $G : \{0,1\}^* \mapsto \mathbb{Z}_q^*$. The long-term trapdoor and hash key pair of an entity is $(TK_l, HK_l) = (x \in_R \mathbb{Z}_q^*, X = \alpha^x \in \mathbb{Z}_p^*)$, and the ephemeral trapdoor and hash key pair of an entity is $(TK_e, HK_e) = (y \in_R \mathbb{Z}_q^*, Y = \alpha^y \in \mathbb{Z}_p^*)$. An entity generates a trapdoor hash of a message $m \in \{0,1\}^*$ using the hash key $HK = (X, Y)$, by choosing an element $r \in_R \mathbb{Z}_q^*$ and computing the hash as $TH_{HK}(m, r) = \alpha^{H(m)}(XY)^r \mod q$. Given system parameters params the trapdoor key $TK = (x, y)$, message $m \in \{0,1\}^*$, $r \in \mathbb{Z}_q^*$ and an additional message $m' (\neq m) \in \{0,1\}^*$, an entity computes a collision as follows:

1. Choose $k' \in_R \mathbb{Z}_q^*$ and compute $r' = \alpha^{k'} \mod q$

2. Generate a ephemeral trapdoor key y' as $y' = {r'}^{-1}(H(m) - H(m') + (x + y)r) - x \mod q$, compute the ephemeral hash key $Y' = \alpha^{y'} \mod p$, and generate the trapdoor hash value $TH_{HK'}(m', r') = \alpha^{H(m') + (x + y')r'} \mod q$, where $HK' = (X, Y')$.

3. Solve for t' in $t' = k' - (x + y')G(TH_{HK'}(m', r')||r') \mod q$ and output $\langle t', r', HK' \rangle$.

Here, $\langle m, r, HK, m', r', HK' \rangle$ is the trapdoor collision tuple, and $\langle t', r' \rangle$ is the signature on $TH_{HK'}(m', r')$ verifiable under XY'. In Section 6.2, we show that forging a collision in DL-DTH without knowledge of the long-term trapdoor key is equivalent to the discrete log problem.

3.3 A Discrete Log-based Signature and Multisignature Scheme

We now provide brief description of the well-known Schnorr signature scheme, along with a multisignature scheme that allows combining the multiple Schnorr signatures on the same message into a single compact multisignature.

The Schnorr Signature Scheme, DL-Schnorr: ElGamal signature scheme [23] and its variants are DL-based signature schemes that have been extensively employed in the literature to build authentication protocols. Popular variants of the ElGamal signature scheme, include the NIST standard DSS [22], and the provably secure Schnorr variant [40]. A digital signature scheme consists of four algorithms — parameter generation ($ParGen$), key generation ($KeyGen$), signature generation ($SigGen$) and signature verification ($SigVer$). As in DL-TH, $ParGen$ generates the system public parameters $\mathsf{params} = \langle p, q, \alpha, H \rangle$ and $KeyGen$ generates the public and private key pair, $(SK, PK) = (x, X = \alpha^x)$. The signature generation and verification algorithms are as follows:

$SigGen$**:** Given a message m, an entity uses its private key $SK = x$ and proceeds as follows:

1. Choose an ephemeral private key $k \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral public key as $r = \alpha^k \mod q$.
2. Solve for the *signature parameter* t in the equation, $t \equiv xH(m||r) + k \mod q$.
3. Output $\sigma = \langle t, r \rangle$ as the resulting signature on message, m

$SigVer$**:** To verify a signature σ on m under public key $PK = X$, an entity proceeds as follows:

1. Parse σ as the tuple $\langle t, r \rangle$ and compute $h = H(m||r)$.
2. Compute $r' = \alpha^t X^{-h} \mod q$.
3. Check whether $r \equiv r' \mod q$. If so, output *Valid*, else output *Invalid*.

Schnorr-based Multisignature Scheme: Micali et al. [34] presented a provably secure Schnorr-based mulitsignature scheme called accountable subgroup multisignatures (ASM). ASM is based on ideas from a well-known discrete log-based multisignature construction technique first introduced by Harn et al. [24, 27]. Let $\{E_1, E_2, \ldots, E_n\}$ denote the set of nodes with (x_i, X_i) as the (private, public) key pair of node E_i. Given common system parameters $\mathsf{params} = \langle p, q, \alpha, H \rangle$, generation of a multisignature on a common message m requires multiple-communication rounds among participating nodes (three in case of ASM [34]) and proceeds as follows:

1. Each node E_i chooses an ephemeral private key $k_i \in_R \mathbb{Z}_q^*$ and computes the corresponding ephemeral public key as $r_i = \alpha^{k_i} \mod p$.

2. All nodes, $E_1, E_2, \ldots, E_n$ interact with each other to compute a common ephemeral public key $r = \prod_{i=1}^{n} r_i \mod q$.

3. Each node E_i solves for the signature parameter t_i in the equation, $t_i = k_i - x_i H(m||r) \mod q$.

4. All nodes, $E_1, E_2, \ldots, E_n$ interact with each other to compute $t = \sum_{i=1}^{n} t_i \mod q$. The multisignature on m under public keys $X_1, X_2, \ldots, X_n$ is given by $\sigma = \langle t, r \rangle$.

To verify σ on m, an entity first computes $X = \prod_{i=1}^{n} X_i \mod p$ and computes $r' = \alpha^t X^{H(m||r)} \mod q$. If $r \equiv r' \mod q$ then σ is a valid multisignature on m under public keys $X_1, X_2, \ldots, X_n$.

4 Multi-trapdoor Hash Functions

The proposed authenticated cloud storage system is built using the recently proposed concept of a multi-trapdoor hashing scheme that allows multiple nodes to compute a collision between a given trapdoor hash value and the trapdoor hash of their own respective messages using their trapdoor keys. Table 1 presents the notations that we will use throughout the rest of the paper. A multi-trapdoor hashing scheme involves multiple entities that we label as $E_1, \ldots, E_n$ each with their individual hash and trapdoor keys. Hash values are computed using hash keys of all participants on a sequence of messages and individual participants can compute collisions with a given hash value.

Symbol	Interpretation
$\langle x \rangle_{[a,b]}$	$\langle x_a, x_{a+1} \ldots, x_b \rangle$
$\langle x \to x' \rangle_{[a,b] \setminus S}$	$\langle x_a, x_{a+1} \ldots, x_b \rangle$, where $S \subset \{a, \ldots, b\}$ and $\forall i \in S$, $x_i = x'_i$. For example, with $S = \{k, l\}$, we get $\langle x \to x' \rangle_{[a,b] \setminus \{k,l\}} = \langle x_a, \ldots, x_{k-1}, x'_k, x_{k+1}, \ldots, x_{l-1}, x'_l, x_{l+1}, \ldots, x_b \rangle$
$\langle x, y \rangle_{[a,b]}$	$\langle x_a, y_a, x_{a+1}, y_{a+1}, \ldots, x_b, y_b \rangle$
$\langle x \to x', y \to y' \rangle_{[a,b] \setminus S}$	$\langle x_a, y_a, x_{a+1}, y_{a+1}, \ldots, x_b, y_b \rangle$, where $S \subset \{a, \ldots, b\}$ and $\forall i \in S$, $x_i = x'_i$, $y_i = y'_i$. For example, with $S = \{k, l\}$, we get $\langle x \to x', y \to y' \rangle_{[a,b] \setminus \{k,l\}} = \langle x_a, y_a, \ldots, x_{k-1}, y_{k-1}, x'_k, y'_k, x_{k+1}, y_{k+1}, \ldots, x_{l-1}, y_{l-1}, x'_l, y'_l, x_{l+1}, y_{l+1}, \ldots, x_b, y_b \rangle$.
${}_{=}x$	By appearing in any of the notations above, signifies that all quantities $x_a, x_{a+1}, \ldots, x_b$ are equal, and so, we can drop the subscripts. For example, if we have $\langle x, {}_{=}y \rangle_{[a,b]}$, then this denotes $\langle x_a, y, x_{a+1}, y, \ldots, x_b, y \rangle$.

Table 1: Notations. We assume $a \leq k \leq l \leq b$ in the descriptions and examples.

Definition 2 *A multi-trapdoor hashing scheme μTH with $E_1, \ldots, E_n$ as the participating entities, has the following components:*

$ParGen$**:** *With a security parameter 1^k as input, outputs common system public parameters* params*.*

$KeyGen$**:** *With* params *as input, for each entity E_i ($i \in \{1, n\}$), generates its long-term trapdoor and hash key pair (TK^i_l, HK^i_l).*

$EKeyGen$**:** *With* params *and as input, for an entity E_i ($i \in \{1, n\}$), generates its ephemeral trapdoor and hash key pair (TK^i_e, HK^i_e).*

TH**:** *With* params*, tuple of hash keys $\langle HK \rangle_{[1,n]}$, where each $HK_i = (HK^i_l, HK^i_e)$, and tuple of (message, random element) pairs $\langle m, r \rangle_{[1,n]}$ as inputs, outputs the multi-trapdoor hash $TH_{\langle HK \rangle_{[1,n]}}(\langle m, r \rangle_{[1,n]})$.*

$TrapColGen$**:** *With* params*, $TK_i = (TK^i_l, TK^i_e)$, tuple of (message, random element) pairs $\langle m, r \rangle_{[1,n]}$, and an additional message $m'_i \neq m_i$ as inputs, outputs a collision parameter r'_i along with $HK'_i = (HK^i_l, HK^{i\prime}_e)$ such that*

$$TH_{\langle HK \rangle_{[1,n]}}(\langle m, r \rangle_{[1,n]}) = TH_{\langle HK \to HK' \rangle_{[1,n] \setminus \{i\}}}(\langle m \to m', r \to r' \rangle_{[1,n] \setminus \{i\}})$$

We term the process of computing collisions using $TrapColGen$ in the definition above as a *multi-trapdoor collision* between the message comprising of the collection $\langle m_1, \ldots, m_i, \ldots, m_n \rangle$ and the message comprising of the collection $\langle m_1, \ldots, m'_i, \ldots, m_n \rangle$ (or, in other words, a multi-trapdoor collision between $\langle m \rangle_{[1,n]}$ and $\langle m \to m' \rangle_{[1,n] \setminus \{i\}}$). The function TH is part of a multi-trapdoor hash family $\mathcal{TH}$ described by params, where each TH is associated to a hash key tuple $HK_1, \ldots, HK_n$ and each key HK_i represents the pair (HK^i_l, HK^i_e).

A property of a multi-trapdoor hashing scheme is the ability to combine (or accumulate) collisions that are computed by multiple entities into a single collision between the original messages and the new set of messages (knowledge of the trapdoor keys is not necessary when combining collisions). More precisely, given a multi-trapdoor hash value $TH_{\langle HK\rangle_{[1,n]}}(\langle m,r\rangle_{[1,n]})$ (where, $\langle m,r\rangle_{[1,n]}$ contains messages m_i and m_j), a multi-trapdoor collision between messages $\langle m\rangle_{[1,n]}$ and $\langle m\rightarrow m'\rangle_{[1,n]\setminus\{i\}}$ so that $TH_{\langle HK\rangle_{[1,n]}}(\langle m,r\rangle_{[1,n]}) = TH_{\langle HK\rightarrow HK'\rangle_{[1,n]\setminus\{i\}}}(\langle m\rightarrow m', r\rightarrow r'\rangle_{[1,n]\setminus\{i\}})$, and another multi-trapdoor collision between messages $\langle m\rangle_{[1,n]}$ and $\langle m\rightarrow m'\rangle_{[1,n]\setminus\{j\}}$ so that $TH_{\langle HK\rangle_{[1,n]}}(\langle m,r\rangle_{[1,n]}) = TH_{\langle HK\rightarrow HK'\rangle_{[1,n]\setminus\{j\}}}(\langle m\rightarrow m', r\rightarrow r'\rangle_{[1,n]\setminus\{j\}})$, the collisions can be combined to obtain a multi-trapdoor collision between messages $\langle m\rangle_{[1,n]}$ and $\langle m\rightarrow m'\rangle_{[1,n]\setminus\{i,j\}}$, so that $TH_{\langle HK\rangle_{[1,n]}}(\langle m,r\rangle_{[1,n]}) = TH_{\langle HK\rightarrow HK'\rangle_{[1,n]\setminus\{i,j\}}}(\langle m\rightarrow m', r\rightarrow r'\rangle_{[1,n]\setminus\{i,j\}})$. Similar to conventional trapdoor hash functions, we require that multi-trapdoor hash functions are efficient to compute, and exhibit resistance to collision forgery and key-exposure. Figure 3 shows the operations of a multi-trapdoor hash function. The target trapdoor hash value h is computed over messages $m_1,\ldots,m_n$. Given a message m'_i, each entity E_i uses its trapdoor key TK_i to compute the collision parameter r_i and an ephemeral hash key HK'_i, so that the trapdoor hash of $m_1,\ldots,m'_i,\ldots,m_n$ matches h. After all entities compute collisions with their respective messages, we can accumulate them to obtain a collision between $m_1,\ldots,m_n$ and $m'_1,\ldots,m'_n$.

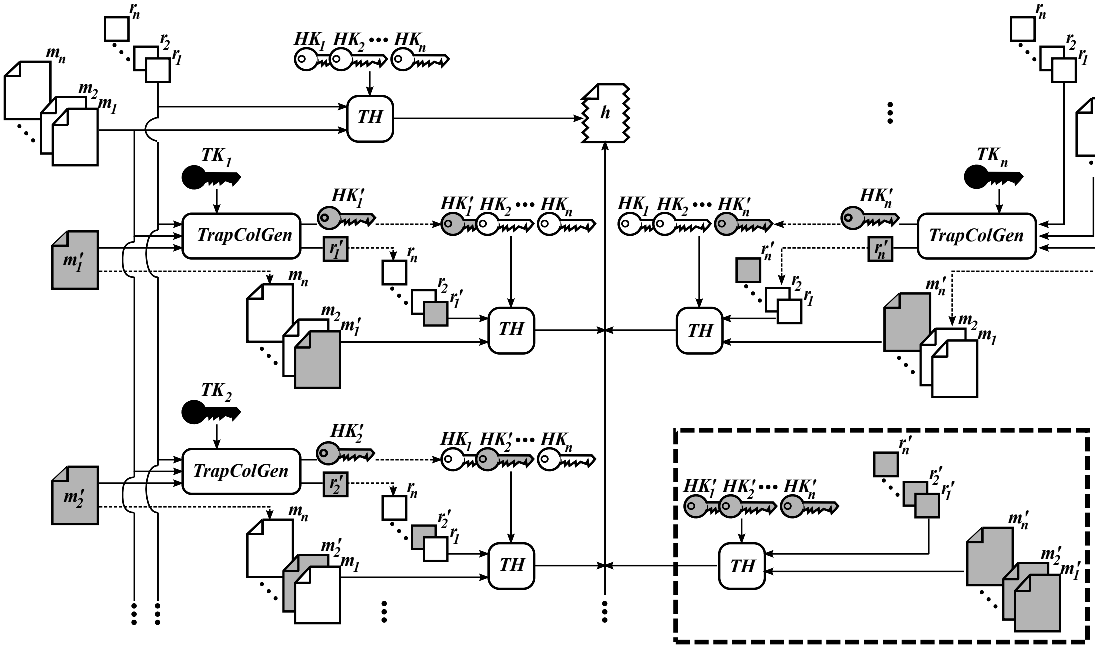


Figure 3: A multi-trapdoor hash function. The boxed region shows the accumulation of collisions that are computed by multiple entities into a single collision between the original messages and the new set of messages.

We now describe a DL-based multi-trapdoor hashing scheme DL-μTH based on the double-trapdoor hashing scheme DL-DTH described in Section 3.2. Similar to DL-DTH, the common system parameters are params $= \langle p,q,\alpha,H,G\rangle$. Also, as in the DL-DTH scheme, the long-term trapdoor and hash key pair of an entity E_i is $(TK^i_l, HK^i_l) = (x_i \in_R \mathbb{Z}^*_q, X_i = \alpha^{x_i} \mod p)$, and the one-time key pair is $(TK^i_e, HK^i_e) = (y_i \in_R \mathbb{Z}^*_q, Y_i = \alpha^{y_i} \mod p)$. Given params, tuple of hash keys $\langle HK\rangle_{[1,n]}$, where each $HK_i = (X_i, Y_i)$, and tuple of (message, random element) pairs $\langle m,r\rangle_{[1,n]}$ as inputs, the multi-trapdoor hash is computed as $TH_{\langle HK\rangle_{[1,n]}}(\langle m,r\rangle_{[1,n]}) = \alpha^{\sum_{i=1}^{n} H(m_i)} \prod_{i=1}^{n} (X_i Y_i)^{r_i} \mod q$. With params, $TK_i = (x_i, y_i)$, tuple of (message, random element) pairs $\langle m,r\rangle_{[1,n]}$, and an additional message $m'_i \neq m_i$ as inputs, an entity computes a collision as follows:

1. Choose $k'_i \in_R \mathbb{Z}^*_q$, compute $r'_i = \alpha^{k'_i} \mod q$, and interact with remaining entities to compute the common parameter $r' = \prod_{i=1}^{n} r'_i \mod q$.

2. Generate an ephemeral trapdoor key y'_i by solving $y'_i = {r'_i}^{-1}(H(m_i) - H(m'_i) + (x_i + y_i)r_i) - x_i \mod q$, compute the ephemeral hash key $Y'_i = \alpha^{y'_i} \mod p$, and generate the trapdoor hash value $h'_i =$

$TH_{\langle HK \to HK' \rangle_{[1,n]\setminus\{i\}}}(\langle m \to m', r \to r' \rangle_{[1,n]\setminus\{i\}}) = \alpha^{\sum_{j=1,j\neq i}^{n} H(m_j)+H(m'_i)} \prod_{j=1,j\neq i}^{n} (X_jY_j)^{r_j}(X_iY'_i)^{r'_i} \mod q$, where $HK'_i = (X_i, Y'_i)$.

3. Solve for t_i in $t_i = k'_i - (x_i + y'_i)G(h'_i||r') \mod q$ and output $\langle t_i, r'_i, HK'_i \rangle$.

After all entities compute their collisions, we obtain n equalities, each representing a collision between trapdoor hash of a new message and the trapdoor hash of the original message, as follows

$$\begin{aligned}
\alpha^{\sum_{i=1}^{n} H(m_i)} \prod_{i=1}^{n} (X_iY_i)^{r_i} &= \alpha^{\sum_{i=2}^{n} H(m_i)+H(m'_1)} \prod_{i=2}^{n} (X_iY_i)^{r_i}(X_1Y'_1)^{r'_1} \\
&= \alpha^{\sum_{i=1,i\neq 2}^{n} H(m_i)+H(m'_2)} \prod_{i=1,i\neq 2}^{n} (X_iY_i)^{r_i}(X_2Y'_2)^{r'_2} \\
&\vdots \\
&= \alpha^{\sum_{i=1}^{n-1} H(m_i)+H(m'_n)} \prod_{i=1}^{n-1} (X_iY_i)^{r_i}(X_nY'_n)^{r'_n}
\end{aligned}$$

We also obtain n signatures $\langle t_1, r'_1 \rangle, \ldots, \langle t_n, r'_n \rangle$ on $h'_1, \ldots, h'_n$ respectively, with all h'_i's being equal to $TH_{\langle HK \rangle_{[1,n]}}(\langle m, r \rangle_{[1,n]})$ as seen above.

These individual collisions can now be *accumulated* together to obtain a multi-trapdoor collision between the original message $\langle m \rangle_{[1,n]}$ and the collection of new messages $(m'_1, \ldots, m'_n)$. For instance, suppose we have collisions between $\langle m \rangle_{[1,n]}$ and the hashes of $\langle m \to m' \rangle_{[1,n]\setminus\{k\}}$ and $\langle m \to m' \rangle_{[1,n]\setminus\{l\}}$, individually. These two collisions can combined into a collision between the trapdoor hash of $\langle m \rangle_{[1,n]}$ and the trapdoor hash of the collection of messages $\langle m \to m' \rangle_{[1,n]\setminus\{k,l\}}$ containing both m'_k and m'_l as follows:

$$\begin{aligned}
TH_{\langle HK \rangle_{[1,n]}}(\langle m, r \rangle_{[1,n]}) &= \alpha^{\sum_{i=1}^{n} H(m_i)} \prod_{i=1}^{n} (X_iY_i)^{r_i} \\
&= \alpha^{H(m'_k)+H(m'_l)+\sum_{i=1,i\neq\{k,l\}}^{n} H(m_i)} (X_kY'_k)^{r'_k}(X_lY'_l)^{r'_l} \prod_{i=1,i\neq\{k,l\}}^{n} (X_iY_i)^{r_i} \\
&= TH_{\langle HK \to HK' \rangle_{[1,n]\setminus\{k,l\}}}(\langle m \to m', r \to r' \rangle_{[1,n]\setminus\{k.l\}})
\end{aligned}$$

Continuing this way, we can combine all individual collisions into a single collision between the trapdoor hash of $\langle m \rangle_{[1,n]}$ and $\langle m' \rangle_{[1,n]}$ to get

$$\begin{aligned}
TH_{\langle HK \rangle_{[1,n]}}(\langle m, r \rangle_{[1,n]}) &= \alpha^{\sum_{i=1}^{n} H(m_i)} \prod_{i=1}^{n} (X_iY_i)^{r_i} \\
&= \alpha^{\sum_{i=1}^{n} H(m'_i)} \prod_{i=1}^{n} (X_iY'_i)^{r'_i} \\
&= TH_{\langle HK' \rangle_{[1,n]}}(\langle m', r' \rangle_{[1,n]})
\end{aligned}$$

To combine the signature components, one simply computes $t = \sum_{i=1}^{n} t_i \mod q$, to get the multisignature $\langle t, r' \rangle$ on h verifiable under the public key $XY' = \prod_{i=1}^{n}(X_iY'_i) \mod p$.

5 Authenticated Cloud Storage System

We now present our authenticated cloud storage system with support for multiple data sources. The system model consists of multiple data sources, a cloud storage service provider (CSSP), an authentication service provider (ASP) and multiple clients. A data source is an entity that originates the files to be stored on the cloud. The CSSP maintains a storage cloud that hosts the database of files generated by the various data sources, and makes the data available to subscribing clients. Such large-scale storage systems are complex and vulnerable to various threats – both malicious and accidental – that can cause data loss or corruption. The ASP is a reliable, trusted and independent third-party entity, with no incentive to act maliciously on its own or by colluding with other entities in the system. The role of the ASP is to ensure that the data sent to a client in response to the client's query is authentic with respect to the data sources, and without loss of integrity. We assume secure and authenticated channels between the data sources, ASP and CSSP, but communication between clients and CSSP need not be secure. The design goals of the proposed authenticated cloud storage system are to provide lightweight operation and high security. In terms of operational costs, our aim is to minimize the storage, computation and communication overhead at each entity, using cryptographic techniques that scale with the number of data sources, and sizes of the database and query response.

The proposed scheme is divided into three phases, namely, initialization, storage and query. In the initialization phase, the data sources, CSSP and ASP agree on the common system parameters and compute the initial cryptographic parameters that will be later used to authenticate the data. During the storage phase, the data sources populate the database, and collaborate with the ASP to generate the authentication tag associated to each piece of data and store them at the database. Finally, during the query phase, the client communicates with the ASP and the CSSP to retrieve the desired data along with the relevant authentication tags, and verify the data's authenticity and integrity. We now describe the three phases in more detail.

Initialization Phase: The initialization phase begins with all entities choosing and agreeing on the common system public parameters $\mathsf{params} = \langle p, q, \alpha, G, H \rangle$, where p, q, α, G, and H are as described in Section 3.2 for the $\mathtt{DL\text{-}DTH}$ scheme. Each data source $E_i (0 \leq i \leq n)$ possesses a trapdoor key $TK_i = (x_i, y_i)$, where $x_i, y_i \in_R \mathbb{Z}_q^*$. The corresponding hash key $HK_i = (X_i, Y_i)$, where $X_i = \alpha^{x_i} \mod p$ and $Y_i = \alpha^{y_i} \mod p$, is published in a publicly available directory. Next, the data sources and ASP perform the following operations:

1. Each E_i chooses $k_i \in_R \mathbb{Z}_q^*$, computes $r_i = \alpha^{k_i} \mod q$, and sends $\langle r_i, HK_i \rangle$ to the ASP.
2. After receiving r_i from each data source E_i, the ASP does the following:
   (a) Assembles $\bar{m}$ as some arbitrary, but meaningful, message pertaining to the data sources (e.g., their identities), and chooses $\bar{r} \in_R \mathbb{Z}_q^*$.
   (b) Computes the product $r = \prod_{i=1}^{n} r_i \mod q$, and broadcasts $\langle \bar{m}, \bar{r}, r \rangle$ to all the data sources $E_1, \ldots, E_n$.

The ASP stores $\bar{m}$, $\bar{r}$, and $r_1, \ldots, r_n$. In the storage phase, the data sources use the tuple $\langle \bar{m}, \bar{r} \rangle$ and their trapdoor keys to compute collisions between the common trapdoor hash value $h = TH_{\langle HK \rangle_{[1,n]}}(\langle {}_{=}\bar{m}, {}_{=}\bar{r} \rangle_{[1,n]}) = \alpha^{nH(\bar{m})} (\prod_{i=1}^{n} X_i Y_i)^{\bar{r}} \mod q$, where $\langle {}_{=}\bar{m}, {}_{=}\bar{r} \rangle_{[1,n]}$ denotes $\langle \underbrace{\bar{m}, \bar{r}, \bar{m}, \bar{r}, \ldots, \bar{m}, \bar{r}}_{n \text{ pairs}} \rangle$, and the trapdoor hashes of their respective data files that they wish to store with the CSSP.

Storage Phase: After receiving $\langle \bar{m}, \bar{r}, r \rangle$, a data source E_i interacts with the ASP to generate the authentication tag for its first piece of data m_{i1} that it wishes to store with the CSSP as follows:

1. Each E_i generates an ephemeral trapdoor key z_{i1} as $z_{i1} = r^{-1}(H(\bar{m}) - H(m_{i1}) + (x_i + y_i)\bar{r}) - x_i \mod q$ and computes the ephemeral hash key $Z_{i1} = \alpha^{z_{i1}} \mod p$.
2. Next it generates the trapdoor hash value $h_{i1} = TH_{\langle HK \rightarrow HK_{\_1} \rangle_{[1,n] \setminus \{i\}}}(\langle {}_{=}\bar{m} \rightarrow m_{\_1}, {}_{=}\bar{r} \rightarrow {}_{=}r \rangle_{[1,n] \setminus \{i\}}) = \alpha^{H(m_{i1}) + (n-1)H(\bar{m})} (X_i Z_{i1})^r (\prod_{j=1, j \neq i}^{n} X_j Y_j)^{\bar{r}} \mod q$, where $\langle HK \rightarrow HK_{\_1} \rangle_{[1,n] \setminus \{i\}}$ denotes $\langle HK_1, \ldots, HK_{i-1}, HK_{i1}, \ldots, HK_n \rangle$, $HK_{i1} = (X_i, Z_{i1})$, and $\langle {}_{=}\bar{m} \rightarrow m_{\_1}, {}_{=}\bar{r} \rightarrow {}_{=}r \rangle_{[1,n] \setminus \{i\}}$ denotes the tuple $\langle \underbrace{\bar{m}, \bar{r}, \bar{m}, \bar{r}, \ldots,}_{i-1 \text{ pairs}} m_{i1}, r, \underbrace{\ldots, \bar{m}, \bar{r}}_{n-i \text{ pairs}} \rangle$.
3. Finally, E_i solves for t_{i1} in $t_{i1} = k_i - (x_i + z_{i1}) G(h_{i1} || r) \mod q$ and sets $a_{i1} = \langle t_{i1}, Z_{i1} \rangle$ as the authentication tag for the data file m_{i1}.

Notice that the trapdoor hash values h and h_{i1} are equal, and represent a multi-trapdoor collision between the messages $\langle {}_{=}\bar{m}\rangle_{[1,n]}$ and $\langle {}_{=}\bar{m} \rightarrow m_{\_1}\rangle_{[1,n]\setminus\{i\}}$. Next, each data source E_i sends the data file m_{i1} to the CSSP for storage, and sends the corresponding authentication tag a_{i1} to the ASP. We assume that given an index $i1$, the ASP can identify and retrieve the authentication tag a_{i1} corresponding to m_{i1}, and the CSSP can retrieve m_{i1} corresponding to a_{i1}. This can be easily achieved by embedding some auxiliary indexing information into m_{i1} and a_{i1}. To generate the authentication tag for a subsequent data file m_{i2} from the E_i, the data source follows Step 1 to compute (z_{i2}, Z_{i2}) so that $h_{i2} = TH_{\langle HK \rightarrow HK_{\_2}\rangle_{[1,n]\setminus\{i\}}}(\langle {}_{=}\bar{m} \rightarrow m_{\_2}, {}_{=}\bar{r} \rightarrow {}_{=}r\rangle_{[1,n]\setminus\{i\}})$ – where, $HK_{i2} = (X_i, Z_{i2})$ – equals the common trapdoor hash value h. Since $h = h_{i1} = h_{i2}$, E_i need not generate h_{i2} in Step 2, and instead, directly generates the signature component t_{i2} as $t_{i2} = k_i - (x_i + z_{i2})G(h_{i1}||r)$ mod q. Finally, E_i sets $a_{i2} = \langle t_{i2}, Z_{i2}\rangle$ as the authentication tag for the data file m_{i2}.

The value r used in computing the trapdoor hash collision between h and h_{i1}, h_{i2}, etc., and in the computation of t_{i1}, t_{i2}, etc., does not change from one data file to the next. Even though, this is a departure from the conventional technique for generating trapdoor collisions and Schnorr signatures, in Section 6.2 we prove that this does not effect the security of the scheme. The intuitive reasoning behind maintaining security despite using a fixed r value is that the trapdoor keys used in computing the collision and in generating the Schnorr signature change with every message. Thus, the ephemeral trapdoor keys (rather than r) take on the role of randomizing the collision and signature generation processes.

Query Phase: After all data sources have populated the database at the CSSP, and stored the relevant authentication tags at the ASP, clients can query and retrieve authenticated data by interacting with the CSSP and the ASP. When a client wishes to query the cloud storage service, it initiates and establishes connections with both the CSSP and ASP. The client begins with sending its query to the CSSP. The CSSP retrieves the data matching the query, sends the indices to the ASP, and the data to the client. The ASP then retrieves the relevant authentication tags matching the query response, combines them into a single compact value, and sends the aggregate authentication tag to the client. Finally, the client authenticates the data using the aggregate authentication tag.

For simplicity, assume the query response consists of the messages $\langle m_{11}, \ldots, m_{1e_1}\rangle$, $\langle m_{21}, \ldots, m_{2e_2}\rangle$, $\ldots$, $\langle m_{l1}, \ldots, m_{le_l}\rangle$, where e_i represents the number of data files from source E_i in the query response, and $l \leq n$ represents the number of sources whose files are included in the response. Let $e = \max_{1\leq i\leq l}(e_i)$. The ASP generates the aggregate authentication tag as follows:

1. For each m_{ie_i}, $1 \leq i \leq l$, retrieve the authentication tags $a_{ie_i} = \langle t_{ie_i}, Z_{ie_i}\rangle$.
2. For $1 \leq i \leq l$ and $e_i < j \leq e$, set $m_{ij} = m_{ie_i}$, $Z_{ij} = Z_{ie_i}$ and $t_{ij} = t_{ie_i}$.
3. Compute $t_a = \sum_{i=1}^{l}(\sum_{j=1}^{e} t_{ij}) \mod q$, $Z_a = \prod_{i=1}^{l}(\prod_{j=1}^{e} Z_{ij}) \mod p$, and $r_a = r(\prod_{i=l+1}^{n} r_i)^{-1}$ mod q, where $r = \prod_{i=1}^{n} r_i \mod q$ as computed during the initialization phase.
4. Assemble the aggregate authentication tag as $a = \langle t_a, r_a, Z_a, e, \bar{r}\rangle$.
5. Choose $k_{asp} \in_r \mathbb{Z}_q^*$, compute $r_{asp} = \alpha^{k_{asp}} \mod q$, and solve for t_{asp} in $t_{asp} = k_{asp} - x_{asp}G(a||\bar{m}||r_{asp})$ mod q to obtain $\sigma_{asp} = \langle t_{asp}, r_{asp}\rangle$.
6. Send $\langle a, \sigma_{asp}, \bar{m}\rangle$ to the client.

After receiving $\langle a, \sigma_{asp}, \bar{m}\rangle$ from the ASP and the query response $\langle m_{11}, \ldots, m_{1e_1}\rangle$, $\langle m_{21}, \ldots, m_{2e_2}\rangle$, $\ldots$, $\langle m_{l1}, \ldots, m_{le_l}\rangle$ from the CSSP, the client verifies the authenticity of the response as follows:

1. Compute $r'_{asp} = \alpha^{t_{asp}} X_{asp}^{G(a||\bar{m}||r_{asp})} \mod q$ and check whether $r_{asp} \equiv r'_{asp} \mod q$. If not, output *Invalid* and halt.
2. Compute $X_a = \prod_{i=1}^{l} X_i \mod p$, $h_l = TH_{\langle HK\rangle_{[1,l]}}(\langle {}_{=}\bar{m}, {}_{=}\bar{r}\rangle_{[1,l]}) = \alpha^{lH(\bar{m})}(\prod_{i=1}^{l} X_i Y_i)^{\bar{r}} \mod q$, and $h = TH_{\langle HK\rangle_{[1,n]}}(\langle {}_{=}\bar{m}, {}_{=}\bar{r}\rangle_{[1,n]}) = \alpha^{nH(\bar{m})}(\prod_{i=1}^{n} X_i Y_i)^{\bar{r}} \mod q$.

3. For $1 \leq i \leq l$ and $e_i < j \leq e$, set $m_{ij} = m_{ie_i}$ and compute $h_m = TH_{\langle HK' \rangle_{[1,l]}}(\langle m, {\scriptscriptstyle =}r \rangle_{[1,l]}) = \alpha^{\sum_{i=1}^{l}(\sum_{j=1}^{e} H(m_{ij}))}((X_a)^e Z_a)^r \mod q$, where $m_i = \langle m_{i1}, \ldots, m_{ie} \rangle$, and $HK'_i = \langle HK_{i1}, \ldots, HK_{ie} \rangle$ with each $HK_{ij} = (X_i, Z_{ij})$.
4. Check whether $(h_l)^e \equiv h_m \mod q$. If not, output $Invalid$ and halt.
5. Compute $r'_a = \alpha^{t_a}((X_a)^e Z_a)^{G(h||r)} \mod q$, and check whether $r_a^e \equiv r'_a \mod q$. If so, output $Valid$.

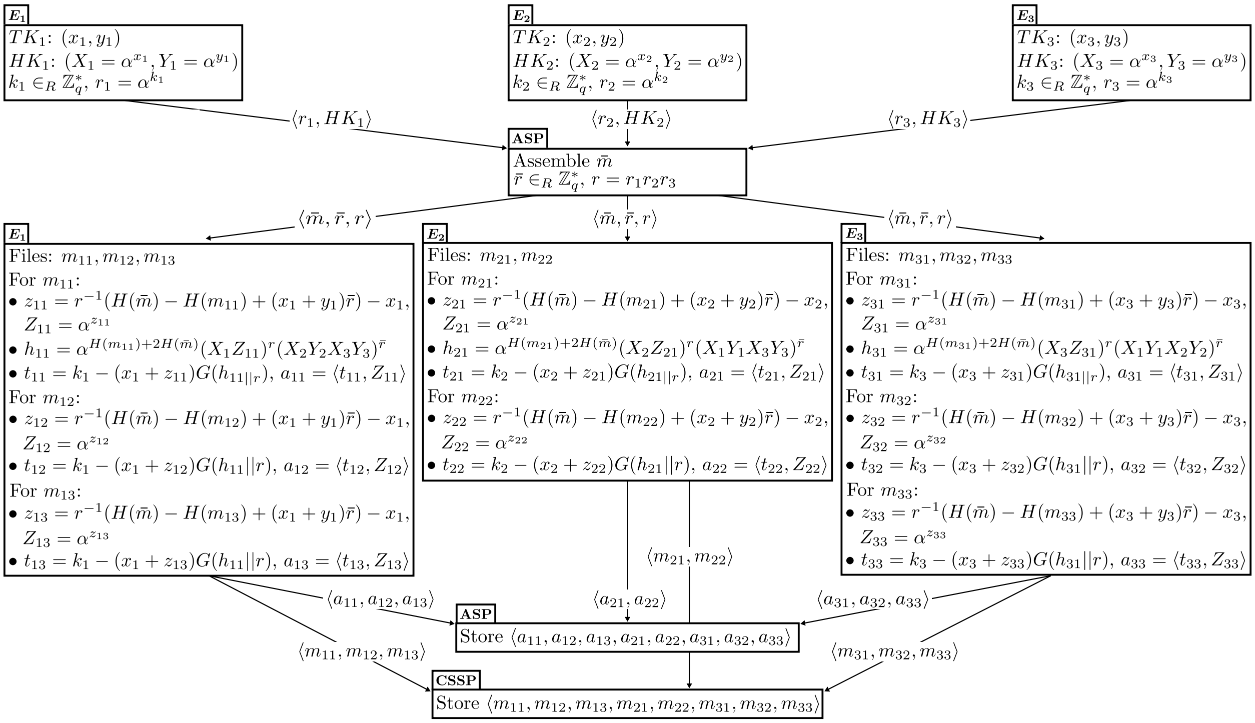
By design, the proposed scheme allows any subset of a designated group of data sources (in the aforementioned case, $E_1, \ldots, E_n$) to participate in the scheme. The requirement of explicitly specifying a group of data sources provides additional security guarantees. Any source that is not in the designated group cannot participate in the scheme, which avoids attacks where an adversarial source is able to convince other entities of being a legitimate participant and insert malicious data that is authenticated using the adversarial source's key. While such an attack does not allow an adversary to modify data content of other participants, however, it can lead to future disruption caused by data inserted by the adversary.

6 Analysis of the Authenticated Cloud Storage Scheme

In this section, we analyze the proposed authenticated cloud storage scheme in terms of its correctness, security and performance.

6.1 Correctness

We now describe an example with three data sources E_1, E_2, E_3 that demonstrates the correctness of the scheme. Figure 4 shows the initialization and storage phases, and Figure 5 shows the query phase.



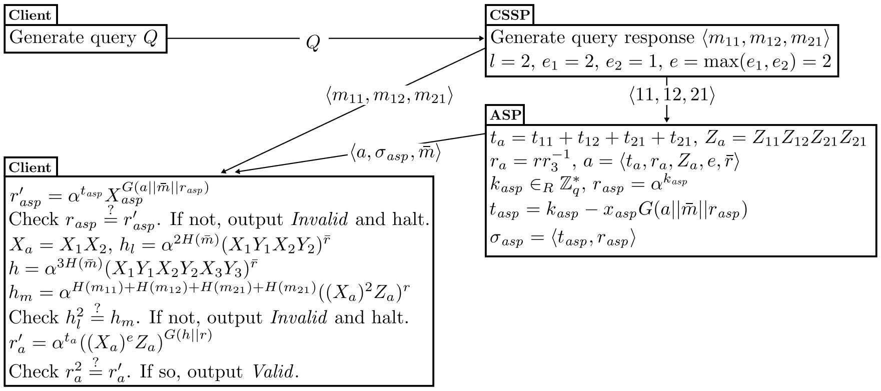
E_1
TK_1: (x_1, y_1)
HK_1: $(X_1 = \alpha^{x_1}, Y_1 = \alpha^{y_1})$
$k_1 \in_R \mathbb{Z}_q^*$, $r_1 = \alpha^{k_1}$
E_2
TK_2: (x_2, y_2)
HK_2: $(X_2 = \alpha^{x_2}, Y_2 = \alpha^{y_2})$
$k_2 \in_R \mathbb{Z}_q^*$, $r_2 = \alpha^{k_2}$
E_3
TK_3: (x_3, y_3)
HK_3: $(X_3 = \alpha^{x_3}, Y_3 = \alpha^{y_3})$
$k_3 \in_R \mathbb{Z}_q^*$, $r_3 = \alpha^{k_3}$
$\langle r_1, HK_1 \rangle$
$\langle r_2, HK_2 \rangle$
$\langle r_3, HK_3 \rangle$
ASP
Assemble $\bar{m}$
$\bar{r} \in_R \mathbb{Z}_q^*$, $r = r_1 r_2 r_3$
$\langle \bar{m}, \bar{r}, r \rangle$
$\langle \bar{m}, \bar{r}, r \rangle$
$\langle \bar{m}, \bar{r}, r \rangle$
E_1
Files: m_{11}, m_{12}, m_{13}
For m_{11}:
• $z_{11} = r^{-1}(H(\bar{m}) - H(m_{11}) + (x_1 + y_1)\bar{r}) - x_1$, $Z_{11} = \alpha^{z_{11}}$
• $h_{11} = \alpha^{H(m_{11}) + 2H(\bar{m})}(X_1 Z_{11})^r (X_2 Y_2 X_3 Y_3)^{\bar{r}}$
• $t_{11} = k_1 - (x_1 + z_{11})G(h_{11||r})$, $a_{11} = \langle t_{11}, Z_{11} \rangle$
For m_{12}:
• $z_{12} = r^{-1}(H(\bar{m}) - H(m_{12}) + (x_1 + y_1)\bar{r}) - x_1$, $Z_{12} = \alpha^{z_{12}}$
• $t_{12} = k_1 - (x_1 + z_{12})G(h_{11}||r)$, $a_{12} = \langle t_{12}, Z_{12} \rangle$
For m_{13}:
• $z_{13} = r^{-1}(H(\bar{m}) - H(m_{13}) + (x_1 + y_1)\bar{r}) - x_1$, $Z_{13} = \alpha^{z_{13}}$
• $t_{13} = k_1 - (x_1 + z_{13})G(h_{11}||r)$, $a_{13} = \langle t_{13}, Z_{13} \rangle$
E_2
Files: m_{21}, m_{22}
For m_{21}:
• $z_{21} = r^{-1}(H(\bar{m}) - H(m_{21}) + (x_2 + y_2)\bar{r}) - x_2$, $Z_{21} = \alpha^{z_{21}}$
• $h_{21} = \alpha^{H(m_{21}) + 2H(\bar{m})}(X_2 Z_{21})^r (X_1 Y_1 X_3 Y_3)^{\bar{r}}$
• $t_{21} = k_2 - (x_2 + z_{21})G(h_{21||r})$, $a_{21} = \langle t_{21}, Z_{21} \rangle$
For m_{22}:
• $z_{22} = r^{-1}(H(\bar{m}) - H(m_{22}) + (x_2 + y_2)\bar{r}) - x_2$, $Z_{22} = \alpha^{z_{22}}$
• $t_{22} = k_2 - (x_2 + z_{22})G(h_{21}||r)$, $a_{22} = \langle t_{22}, Z_{22} \rangle$
E_3
Files: m_{31}, m_{32}, m_{33}
For m_{31}:
• $z_{31} = r^{-1}(H(\bar{m}) - H(m_{31}) + (x_3 + y_3)\bar{r}) - x_3$, $Z_{31} = \alpha^{z_{31}}$
• $h_{31} = \alpha^{H(m_{31}) + 2H(\bar{m})}(X_3 Z_{31})^r (X_1 Y_1 X_2 Y_2)^{\bar{r}}$
• $t_{31} = k_3 - (x_3 + z_{31})G(h_{31||r})$, $a_{31} = \langle t_{31}, Z_{31} \rangle$
For m_{32}:
• $z_{32} = r^{-1}(H(\bar{m}) - H(m_{32}) + (x_3 + y_3)\bar{r}) - x_3$, $Z_{32} = \alpha^{z_{32}}$
• $t_{32} = k_3 - (x_3 + z_{32})G(h_{31}||r)$, $a_{32} = \langle t_{32}, Z_{32} \rangle$
For m_{33}:
• $z_{33} = r^{-1}(H(\bar{m}) - H(m_{33}) + (x_3 + y_3)\bar{r}) - x_3$, $Z_{33} = \alpha^{z_{33}}$
• $t_{33} = k_3 - (x_3 + z_{33})G(h_{31}||r)$, $a_{33} = \langle t_{33}, Z_{33} \rangle$
$\langle m_{21}, m_{22} \rangle$
$\langle a_{11}, a_{12}, a_{13} \rangle$
$\langle a_{21}, a_{22} \rangle$
$\langle a_{31}, a_{32}, a_{33} \rangle$
ASP
Store $\langle a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{31}, a_{32}, a_{33} \rangle$
$\langle m_{11}, m_{12}, m_{13} \rangle$
$\langle m_{31}, m_{32}, m_{33} \rangle$
CSSP
Store $\langle m_{11}, m_{12}, m_{13}, m_{21}, m_{22}, m_{31}, m_{32}, m_{33} \rangle$

Figure 4: Initialization and Storage phases with three sources E_1, E_2 and E_3.

After the initialization phase is over each source possesses r, $\bar{m}$ and $\bar{r}$. When E_1 wishes to store a file, say m_{11}, it computes a collision between the common trapdoor hash value $h = TH_{\langle HK\rangle_{[1,3]}}({}_{=}\bar{m}, {}_{=}\bar{r}\rangle_{[1,3]}) = \alpha^{3H(\bar{m})}(X_1Y_1X_2Y_2X_3Y_3)^{\bar{r}}$ and the trapdoor hash of a message containing m_{11}. In particular, E_1 computes (z_{11}, Z_{11}) so that $\alpha^{H(\bar{m})}(X_1Y_1)^{\bar{r}} = \alpha^{H(m_{11})}(X_1Z_{11})^r$. This gives us the multi-trapdoor collision between the messages $\langle \bar{m}, \bar{m}, \bar{m}\rangle$ and $\langle m_{11}, \bar{m}, \bar{m}\rangle$ as follows:

$$
\begin{aligned}
h_{11} &= TH_{\langle HK \rightarrow HK_{\_1}\rangle_{[1,3]\setminus\{1\}}}(\langle {}_{=}\bar{m} \rightarrow m_{\_1}, {}_{=}\bar{r} \rightarrow {}_{=}r\rangle_{[1,3]\setminus\{1\}}) \\
&= \alpha^{H(m_{11})+2H(\bar{m})}(X_1Z_{11})^r(X_2Y_2X_3Y_3)^{\bar{r}} \\
&= \alpha^{H(m_{11})+2H(\bar{m})}\left(X_1\alpha^{(r^{-1}(H(\bar{m})-H(m_{11})+(x_1+y_1)\bar{r})-x_1)}\right)^r(X_2Y_2X_3Y_3)^{\bar{r}} \\
&= \alpha^{H(m_{11})+2H(\bar{m})}\left(X_1^r\alpha^{(H(\bar{m})-H(m_{11})+(x_1+y_1)\bar{r}-x_1r)}\right)(X_2Y_2X_3Y_3)^{\bar{r}} \\
&= \alpha^{3H(\bar{m})}(X_1Y_1X_2Y_2X_3Y_3)^{\bar{r}} \\
&= TH_{\langle HK\rangle_{[1,3]}}({}_{=}\bar{m}, {}_{=}\bar{r}\rangle_{[1,3]}) \\
&= h
\end{aligned}
$$

It is straightforward to show that computing (z_{12}, Z_{12}) and (z_{13}, Z_{13}) as shown in Figure 4, result in the trapdoor hash values h_{12} of $\langle m_{12}, \bar{m}, \bar{m}\rangle$ and h_{13} of $\langle m_{13}, \bar{m}, \bar{m}\rangle$ also being equal to h, i.e., $h = h_{12} = h_{13}$. Similar results follow for the sources E_2 and E_3, and their respective files. Also, note that all the signatures $t_{11}, \ldots, t_{33}$ are generated on the same message, since $h_{11} = h_{21} = h_{31}$.



Client
Generate query Q
Q
CSSP
Generate query response $\langle m_{11}, m_{12}, m_{21}\rangle$
$l = 2$, $e_1 = 2$, $e_2 = 1$, $e = \max(e_1, e_2) = 2$
$\langle m_{11}, m_{12}, m_{21}\rangle$
$\langle 11, 12, 21\rangle$
ASP
$t_a = t_{11} + t_{12} + t_{21} + t_{21}$, $Z_a = Z_{11}Z_{12}Z_{21}Z_{21}$
$r_a = rr_3^{-1}$, $a = \langle t_a, r_a, Z_a, e, \bar{r}\rangle$
$k_{asp} \in_R \mathbb{Z}_q^*$, $r_{asp} = \alpha^{k_{asp}}$
$t_{asp} = k_{asp} - x_{asp}G(a||\bar{m}||r_{asp})$
$\sigma_{asp} = \langle t_{asp}, r_{asp}\rangle$
$\langle a, \sigma_{asp}, \bar{m}\rangle$
Client
$r'_{asp} = \alpha^{t_{asp}}X_{asp}^{G(a||\bar{m}||r_{asp})}$
Check $r_{asp} \overset{?}{=} r'_{asp}$. If not, output *Invalid* and halt.
$X_a = X_1X_2$, $h_l = \alpha^{2H(\bar{m})}(X_1Y_1X_2Y_2)^{\bar{r}}$
$h = \alpha^{3H(\bar{m})}(X_1Y_1X_2Y_2X_3Y_3)^{\bar{r}}$
$h_m = \alpha^{H(m_{11})+H(m_{12})+H(m_{21})+H(m_{21})}((X_a)^2Z_a)^r$
Check $h_l^2 \overset{?}{=} h_m$. If not, output *Invalid* and halt.
$r'_a = \alpha^{t_a}((X_a)^eZ_a)^{G(h||r)}$
Check $r_a^2 \overset{?}{=} r'_a$. If so, output *Valid*.

Figure 5: Query phase in which client's query generates a response containing messages from E_1 and E_2.

Following the storage phase, assume that a client queries the CSP whose response contains two files, $\langle m_{11}, m_{12}\rangle$, from E_1, and one file, $\langle m_{21}\rangle$, from E_2. The ASP generates the aggregate authentication tag as $a = \langle t_a, r_a, Z_a, e, \bar{r}\rangle$, where $e = 2$, and sends $\langle a, \sigma_{asp}, \bar{m}\rangle$ to the client. To verify the authenticity of the query response, the client first verifies the ASP's signature σ_{asp} on $a||\bar{m}$ under the public key X_{asp}. Next, the client checks whether $h_m = h_l^e$. Since, $\alpha^{H(\bar{m})}(X_1Y_1)^{\bar{r}} = \alpha^{H(m_{11})}(X_1Z_{11})^r = \alpha^{H(m_{12})}(X_1Z_{12})^r$ and $\alpha^{H(\bar{m})}(X_2Y_2)^{\bar{r}} = \alpha^{H(m_{21})}(X_2Z_{21})^r$, we have:

$$
\begin{aligned}
h_m &= \alpha^{(H(m_{11})+H(m_{12})+H(m_{21})+H(m_{21}))}(X_a^2 Z_a)^r \\
&= \alpha^{(H(m_{11})+H(m_{12})+H(m_{21})+H(m_{21}))}(X_1 Z_{11} X_1 Z_{12} X_2 Z_{21} X_2 Z_{21})^r \\
&= \alpha^{(H(\bar{m})+H(\bar{m})+H(\bar{m})+H(\bar{m}))}(X_1 Y_1 X_1 Y_1 X_2 Y_2 X_2 Y_2)^{\bar{r}} \\
&= \left(\alpha^{2H(\bar{m})}(X_1 Y_1 X_2 Y_2)^{\bar{r}}\right)^2 \\
&= h_l^2
\end{aligned}
$$

Finally, the client checks the validity of the multisignature $\langle t_a, r_a\rangle$ on the message h under the public key $X_a^2 Z_a$. We see that:

$$
\begin{aligned}
r'_a &= \alpha^{t_a}((X_a)^2 Z_a)^{G(h||r)} \\
&= \alpha^{(t_{11}+t_{12}+t_{21}+t_{21})}((X_1 X_2)^2 Z_{11} Z_{12} Z_{21} Z_{21})^{G(h||r)} \\
&= r_1(X_1 Z_{11})^{-G(h||r)} r_1(X_1 Z_{12})^{-G(h||r)} r_2(X_2 Z_{21})^{-G(h||r)} r_2(X_2 Z_{21})^{-G(h||r)}((X_1 X_2)^2 Z_{11} Z_{12} Z_{21} Z_{21})^{G(h||r)} \\
&= (r_1 r_2)^2 \\
&= r_a^2
\end{aligned}
$$

6.2 Security Analysis

The proposed multi-trapdoor hashing scheme DL-μTH is based on the double-trapdoor hashing scheme DL-DTH. Thus, collision forgery and key exposure resistance of DL-μTH depends on the DL-DTH scheme's resistance to the same attacks. The difficulty of forging collisions and key exposure in DL-DTH is based on the difficulty of solving the well-known discrete logarithm problem in subgroup of $\mathbb{Z}_p^*$. We begin with proving the following two theorems that establish the security of DL-DTH.

Theorem 1 *The proposed trapdoor hashing scheme* DL-DTH *is collision forgery resistant.*

Proof: We prove the forgery resistance property of the proposed trapdoor hashing scheme by showing that the discrete log problem in subgroup of $\mathbb{Z}_p^*$ reduces to collision forgery, thus violating the well known discrete log assumption.

Assume that there exists a PPT collision forger $\mathcal{F}$ against the proposed trapdoor hashing scheme with non-negligible advantage. Given a hash key $HK = (X, Y)$ and parameters $\langle p, q, \alpha, H, G\rangle$, $\mathcal{F}$ runs in polynomial time and outputs the tuple $\langle m, r, m', r', HK', t'\rangle$, where $HK' = (X, Y')$, such that $m \neq m'$, $r \neq r'$, $h = TH_{HK}(m, r) = TH_{HK'}(m', r')$, and $\alpha^{(t')}(XY')^{G(h||r')} \equiv r' \mod q$ with non-negligible probability. Given $\mathcal{F}$ we can construct a PPT algorithm $\mathcal{D}$ that breaks the discrete log problem as follows. $\mathcal{D}$ is given a DLP instance $\langle p, q, \alpha, X\rangle$. $\mathcal{D}$ needs to find $x \in \mathbb{Z}_q^*$ such that $X = \alpha^x \mod p$. The hash function G behaves as a random oracle $\mathcal{O}_G$ that $\mathcal{D}$ simulates. This means that $\mathcal{D}$ answers any hash queries to $\mathcal{O}_G$ by a random value for each new query [39] (with identical anwers if the same query is asked twice). For instance when $\mathcal{F}$ queries $\mathcal{O}_G$ with $\langle h, r'\rangle$, where $h = TH_{HK}(m, r)$ for some m and r, $\mathcal{O}_G$ returns g if $\exists g$ such that $g = G(h||r')$. Otherwise $\mathcal{D}$ chooses $g \in_R \mathbb{Z}_q^*$, sets $G(h||r') = g$, stores g in the hash entry for $G(h||r')$ and returns g to $\mathcal{F}$.

On input $\langle p, q, \alpha, X\rangle$, $\mathcal{D}$ chooses $y \in_R \mathbb{Z}_q^*$, computes $Y = \alpha^y \mod p$ and sets $HK = (X, Y)$. $\mathcal{D}$ then runs an instance of forger $\mathcal{F}$ with HK as input, answering any hash queries to $\mathcal{O}_G$, until $\mathcal{F}$ produces the collision forgery $\langle m, r, m', r', HK', t'\rangle$, where $HK' = (X, Y')$, $h' = TH_{HK'}(m', r')$ and $\alpha^{t'}(XY')^{G(h'||r')} \equiv r' \mod q$. Let g' be the response $\mathcal{D}$ gave when $\mathcal{F}$ made the query $\langle h', r'\rangle$ to $\mathcal{O}_G$. Using the oracle replay attack [39], $\mathcal{D}$ rewinds $\mathcal{F}$ to the point when $\mathcal{F}$ makes the query, $\langle h', r'\rangle$ to $\mathcal{O}_G$, and gives $\mathcal{F}$ a new randomly chosen value $g'' \neq g' \in_R \mathbb{Z}_q^*$. $\mathcal{D}$ continues execution of $\mathcal{F}$, until $\mathcal{F}$ produces another collision forgery of the form $\langle m, r, m', r', HK', t''\rangle$, where $\alpha^{(t'')}(XY')^{G(h'||r')} \equiv r' \mod q$. Given the two collisions produced by

$\mathcal{F}$, we now have $r' \equiv \alpha^{(t')}(XY')^{(g')} \equiv \alpha^{(t'')}(XY')^{(g'')} \mod p$. $\mathcal{D}$ now computes $sk' = (t' - t'')(g'' - g')^{-1} \mod q$, where $sk' = x + y'$, and y' is the discrete log of Y'. Finally, $\mathcal{D}$ computes the discrete log of X as $x = r^{-1}(H(m') - H(m) + sk'r') - y \mod q$. □

Theorem 2 *The proposed double-trapdoor hashing scheme,* `DL-DTH`*, is key exposure resistant.*

Proof: Key exposure resistance in `DL-DTH` implies that given a trapdoor collision tuple $\langle m, r, HK, m', r', HK' \rangle$ such that $TH_{HK}(m, r) = TH_{HK'}(m', r')$, where $HK = (X, Y)$ and $HK' = (X, Y')$, it is computationally infeasible to find the long-term trapdoor key, x corresponding to X.

Assume that there exists a PPT algorithm $\mathcal{K}$ that succeeds in key exposure against `DL-DTH` with non-negligible advantage. Given $\mathcal{K}$ we can construct a PPT algorithm $\mathcal{D}$ that breaks the discrete log problem. $\mathcal{D}$ is given a DLP instance $\langle p, q, \alpha, X \rangle$ and needs to find $x \in \mathbb{Z}_q^*$ such that $X = \alpha^x \mod p$. $\mathcal{D}$ chooses $y \in_R \mathbb{Z}_q^*$, computes $Y = \alpha^y \mod p$, and sets $HK = (X, Y)$ with the corresponding $TK = (x, y)$. Next, $\mathcal{D}$ chooses $m, m' \in_R \{0,1\}^*$ and $r \in_R \mathbb{Z}_q^*$, and uses the $TrapColGen$ algorithm of `DL-DTH` to generate a double-trapdoor collision tuple $\langle m, r, HK, m', r', HK' \rangle$, where $HK' = (X, Y')$. $\mathcal{D}$ then runs an instance of forger $\mathcal{K}$ with $\langle m, r, HK, m', r', HK' \rangle$ as input. When $\mathcal{K}$ outputs x, $\mathcal{D}$ outputs the same value x as the discrete log of X. This contradicts the well known discrete log assumption. Thus, the proposed double-trapdoor hashing scheme `DL-DTH` is key-exposure free. □

The proposed multi-trapdoor hashing scheme `DL-`μ`TH` uses `DL-DTH` as the underlying trapdoor hash function. Given Theorems 1 and 2, corresponding security results of `DL-`μ`TH` immediately follow. This is because, forging a collision or key exposure in `DL-`μ`TH` requires an adversary to be able to do the same in the underlying `DL-DTH` scheme, which we have shown to be atleast as hard as breaking the discrete log problem. Thus, we have the the following theorem.

Theorem 3 *The proposed multi-trapdoor hashing scheme* `DL-`μ`TH` *is collision forgery and key exposure resistant.*

We now show that the proposed authenticated cloud storage system is secure against adversarial attacks under the discrete log assumption. The security of the proposed scheme relies on the security of its building blocks, namely, the multi-trapdoor hashing scheme, `DL-`μ`TH`, the Schnorr signature scheme [40] and the multisignature scheme by Micali et al. [34].

Theorem 4 *The discrete log problem in subgroup of $\mathbb{Z}_p^*$ reduces to forging the authentication tag associated to a single message or a set of messages (from multiple sources).*

Proof: We begin with considering the forgery of an authentication tag associated to an individual message generated by an entity E_i during the storage phase of the scheme. Assume that there exists a PPT tag forger $\mathcal{S}$ against the storage phase of the proposed authenticated cloud storage system. Given system parameters $\langle p, q, \alpha, H, G \rangle$, tuple of hash keys $\langle HK \rangle_{[1,n]} = \langle (X_1, Y_1), \ldots, (X_n, Y_n) \rangle$, and a tuple $\langle \bar{m}, \bar{r}, r \rangle$ as inputs, $\mathcal{S}$ runs in polynomial time and outputs a forged authentication tag $a_{ij} = \langle t_{ij}, Z_{ij} \rangle$ associated to an arbitrary message m_{ij} (chosen by $\mathcal{S}$) on behalf an entity E_i $(1 \leq i \leq n)$, such that $\alpha^{H(\bar{m})}(X_i Y_i)^{\bar{r}} = \alpha^{H(m_{ij})}(X_i Z_{ij})^r$ and $\alpha^{t_{ij}}(X_i Z_{ij})^{G(h||r)} \equiv r \mod q$, where $h = \alpha^{nH(\bar{m})}(\prod_{i=1}^n X_i Y_i)^{\bar{r}}$, with non-negligible probability. Given $\mathcal{S}$ we can construct a PPT algorithm $\mathcal{D}$ that breaks the discrete log problem as follows. $\mathcal{D}$ is given a DLP instance $\langle p, q, \alpha, X \rangle$, and simulates random oracle $\mathcal{O}_G$ for the hash function G. For $1 \leq i \leq n$, $\mathcal{D}$ chooses $x_i, y_i \in_R \mathbb{Z}_q^*$, computes $Y_i = \alpha^{-y_i} \mod p$, computes $X_i = X^{x_i}(Y_i)^{-1} \mod p$, and sets $HK_i = (X_i, Y_i)$. Next $\mathcal{D}$ chooses $\bar{r}, r \in_R \mathbb{Z}_q^*$, and a random message $\bar{m}$. $\mathcal{D}$ then runs an instance of forger $\mathcal{S}$ with $\langle p, q, \alpha, H, G, \langle HK \rangle_{[1,n]}, \bar{m}, \bar{r}, r \rangle$ as inputs, answering any hash queries to $\mathcal{O}_G$, until $\mathcal{S}$ produces the forged authentication tag $a_{ij} = \langle t_{ij}, Z_{ij} \rangle$. Let g be the response $\mathcal{D}$ gave when $\mathcal{S}$ made the query $\langle h, r \rangle$ to $\mathcal{O}_G$. Using the oracle replay attack [39], $\mathcal{D}$ rewinds $\mathcal{S}$ to the point when $\mathcal{S}$ makes the query, $\langle h, r \rangle$ to $\mathcal{O}_G$, and gives $\mathcal{S}$ a new randomly chosen value $g' \neq g \in_R \mathbb{Z}_q^*$. $\mathcal{D}$ continues execution of $\mathcal{S}$, until $\mathcal{S}$ produces another forgery of the form $a'_{ij} = \langle t'_{ij}, Z_{ij} \rangle$, where $\alpha^{H(\bar{m})}(X_i Y_i)^{\bar{r}} = \alpha^{H(m_{ij})}(X_i Z_{ij})^r$ and $\alpha^{(t'_{ij})}(X_i Z_{ij})^{g'} \equiv r \mod q$. Given the two forged tags produced by $\mathcal{S}$, we now have $r \equiv \alpha^{t_{ij}}(X_i Z_{ij})^g \equiv \alpha^{t'_{ij}}(X_i Z_{ij})^{g'} \mod p$. $\mathcal{D}$ now computes $sk = (t'_{ij} - t_{ij})(g - g')^{-1} \mod q$, where sk is the discrete log of $X_i Z_{ij}$. Finally $\mathcal{D}$ computes the discrete log of X as $x = (x_i \bar{r})^{-1}(H(m_{ij}) - H(\bar{m}) + (sk)r) \mod q$.

Now consider the forgery of an aggregate authentication tag associated with messages from entities $E_1, \ldots, E_n$ during the query phase of the scheme. Assume that there exists a PPT aggregate tag forger $\mathcal{A}$ against the query phase of the proposed authenticated cloud storage system. $\mathcal{A}$ is given system parameters $\langle p, q, \alpha, H, G \rangle$, tuple of hash keys $\langle HK \rangle_{[1,n]} = \langle (X_1, Y_1), \ldots, (X_n, Y_n) \rangle$, and the tuple $\langle \bar{m}, \bar{r}, r \rangle$ as inputs. In addition, $\mathcal{A}$ adaptively queries the entities to generate a list of authentic (message, tag) pairs $\langle (m_{11}, a_{11}), \ldots, (m_{1e_1}, a_{1e_1}) \rangle$, $\langle (m_{21}, a_{21}), \ldots, (m_{2e_2}, a_{2e_2}) \rangle$, $\ldots$, $\langle (m_{n1}, a_{n1}), \ldots, (m_{ne_n}, a_{ne_n}) \rangle$, where e_i represents the number of data files from source E_i, and $a_{ij} = \langle t_{ij}, Z_{ij} \rangle$ is the authentication tag of m_{ij} generated by E_i. $\mathcal{A}$ runs in polynomial time and outputs a forged aggregate authentication tag $a' = \langle t'_a, r'_a, Z'_a, \bar{r} \rangle$ along with a signature $\sigma'_{asp} = \langle t'_{asp}, r'_{asp} \rangle$ corresponding to the messages $m'_1, \ldots, m'_n$, where each m'_i is associated with an entity E_i, and σ'_{asp} is a signature on the message $a' || \bar{m}$ verifiable under the ASP's public key X_{asp}[2]. It's straightforward to show that we can use $\mathcal{A}$ to construct a PPT algorithm $\mathcal{D}$ that breaks the discrete log problem since, the forgery by $\mathcal{A}$ contains a forged ASM multisignature [34] and a forged Schnorr signature [40] that are both shown to be at least as hard as the discrete log problem. □

6.3 Performance

Table 2 shows the performance evaluation results of the proposed authenticated cloud-based storage system. For a security level of 2048-bits, the system parameters p and q are 2048-bits and 224-bits, respectively, and SHA-2 with a 224-bit output is used as the hash functions H and G. Thus, exponents in exponentiation are 224-bits in size.

Operation	**Exponentiation**	**Modular multiplication**	**Hash**	**Modular summation**	**Inversion**
Individual tag generation	4	$O(n)$	$O(1)$	$O(1)$	1
Aggregate tag generation	1	$O(q)$	$O(1)$	$O(q)$	1
Aggregate tag verification	10	$O(n)$	$O(q)$	$O(q)$	–

Table 2: Performance evaluation of authenticated cloud storage system. n—number of data sources, l—number of data sources whose messages are included in a query response, q—number of messages in a query response.

As we can see from Table 2, the number of exponentiations required for generation of individual tags, generation of aggregate tags and verification of aggregate tags remain constant regardless of q (the number of messages in the query response) and l (the number of data sources whose files are included in the response). Although the number of modular multiplications/summations and hash computations grows linearly with the query result size and/or the number of sources, these operations are highly efficient to compute, causing minimal computational overhead even for large values of n, q and l. Note that, in our evaluation, we ignore the cost of exponentiations by e, the maximum number of messages from a single data source, since $e << \lfloor \log_2(q) \rfloor + 1$. In contrast, existing schemes that provide support for multiple sources [36, 37, 38] use aggregate signature scheme BAS by Boneh et al. [6] for generation of individual and aggregate authentication tags, which require much more expensive $l + q + 1$ pairing operations for verification of the aggregate tags. The size of individual and aggregate authentication tags remain (near) constant as well. Assuming a security level of 2048-bits, an individual authentication tag a_{ij} for a file m_{ij} requires 284-bytes of storage. Also, the entire outsourced database requires an additional $(84 + 28n + ((\lfloor \log_2(\bar{m}) \rfloor + 1)/8))$-bytes for storing the values of $\bar{r}$, $\langle r_1, \ldots, r_n \rangle$, and $\bar{m}$. Aggregate authentication tags a that are generated by the ASP during the query phase are a constant 344-bytes long regardless of the query size or number of sources. With the addition

[2] For simplicity, we assume a forgery on query response that contains a single message from each source. More complex forgeries are also possible, but without any improvement to the adversary's advantage.

of σ_{asp}, the total size of authentication information sent by the ASP to the client is a constant 400-bytes. In contrast, BAS-based schemes require only 40-bytes for both the individual and aggregate tags. However, we argue that BAS-based schemes' use of expensive pairing operations and linear growth in the number of pairings for verifying aggregate tags, significantly undermine the benefits of smaller tags. Thus, overall, the proposed scheme achieves superior scalability and efficiency, compared with existing schemes supporting multiple data sources.

7 Conclusions and Future Work

In this paper, we studied the problem of ensuring integrity and authenticity of data in cloud-based storage systems. We found that prior solutions to this problem do not scale well when data in the cloud is populated by multiple sources. To address this issue, we developed a scalable and efficient authenticated cloud storage system with multiple data sources based on multi-trapdoor hash functions. The proposed scheme allows clients to verify the integrity and authenticity of query results from a cloud storage system that is not necessarily trusted and is storing data from multiple sources. The efficiency and scalability is achieved through properties of multi-trapdoor hashes that allows multiple authentication tags that are associated to the files in a query result to be aggregated in a single tag. Our performance analysis showed that the computation cost associated with generating and verifying both the individual and aggregate tags remain (near) constant regardless of the number of files in a query response or the number of data sources; the portion of computation cost that increases linearly are associated with trivial operations like multiplication, hashing, and addition. Also, the size of the aggregate authentication tag remains constant. When compared to other schemes that support multiple sources, we find that, although the size of individual and aggregate authentication tags are smaller, the cost of verifying the aggregate authentication tag increases linearly with the number of sources, which can result in significant drain on client resources. Finally, in our security analysis, we showed that forging a authentication tag associated to a single file or a set of files is at least as hard as solving the well-known discrete log problem.

Currently, the proposed scheme only ensures the authenticity and integrity (or, the correctness) of selection query results. We plan to extend our mechanism to also provide completeness guarantees, which will allow the client to verify that the cloud returns every file that satisfies the query condition [37]. The mechanism will also be augmented to support different types of queries in addition to selection. Three limitations of the proposed scheme are (1) all potential data owners must be known, (2) need for multiple communication rounds between data owners and authentication service provider during storage phase, and (3) need for a trusted authentication service provider. These limitations stem from the use of the proposed `DL-DTH` scheme in the construction of the multi-trapdoor hashing scheme. Our investigations reveal that other existing trapdoor hashing schemes are ill-suited for building multi-trapdoor hash functions as they cause a linear increase in computation cost during collision accumulation; this, in turn, results in a linear increase in verification cost of the aggregate authentication tag. We are currently conducting research on the development of new key exposure-free trapdoor hash functions that are better suited for building multi-trapdoor hash functions.

References

[1] The health information technology for economic and clinical health act. Public Law 111-5, 111th Congress, Feb. 17 2009. Available from `http://www.gpo.gov/fdsys/pkg/PLAW-111publ5/pdf/PLAW-111publ5.pdf`, Accessed Nov 25, 2014.

[2] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Transactions on Information and System Security*, 14(1):12, 2011.

[3] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Proceedings*

of ESORICS, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, volume 3679 of *LNCS*, pages 159–177. Springer, 2005.

[4] Giuseppe Ateniese and Breno de Medeiros. Identity-based chameleon hash and applications. In *Proceedings of FC, 8th International Conference on Financial Cryptography*, volume 3110 of *LNCS*, pages 164–180. Springer, 2004.

[5] Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In *Proceedings of SCN, 4th International Conference on Security in Communication Networks*, volume 3352 of *LNCS*, pages 165–179. Springer, 2004.

[6] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Proceedings of EUROCRYPT: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.

[7] Vason Bourne. Microsoft SMB hosted IT commentary report 140110, February 2010. Available from `http://download.microsoft.com/download/4/1/C/41C365D5-6E3D-4293-A14F-66F16553D6F1/Microsoft_SMB_Hosted_IT_Commentary_Report_140110.pdf`, Accessed December 4, 2012.

[8] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

[9] Santosh Chandrasekhar, Saikat Chakrabarti, and Mukesh Singhal. A trapdoor hash-based mechanism for stream authentication. *IEEE Transactions on Dependable and Secure Computing*, 9(5):699–713, 2012.

[10] Santosh Chandrasekhar, Saikat Chakrabarti, Mukesh Singhal, and Kenneth L. Calvert. Efficient proxy signatures based on trapdoor hash functions. *IET Information Security, Special Issue: Selected papers on multi-agent and distributed information security*, 4(4):322–332, December 2010.

[11] Santosh Chandrasekhar and Mukesh Singhal. Multi-trapdoor hash functions and their applications in network security. In *Proceedings of IEEE-CNS, Second IEEE Conference on Communications and Network Security, San Francisco, California, San Francisco, California, USA, October 29-31*, pages 472–480. IEEE, 2014.

[12] Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. Chameleon hashing without key exposure. In Kan Zhang and Yuliang Zheng, editors, *Proceedings of ISC, Information Security, 7th International Conference, Palo Alto, CA, USA, September 27-29*, volume 3225 of *LNCS*, pages 87–98. Springer, 2004.

[13] Xiaofeng Chen, Fangguo Zhang, Willy Susilo, and Yi Mu. Efficient generic on-line/off-line signatures without key exposure. In Jonathan Katz and Moti Yung, editors, *Proceedingsw of ACNS, Applied Cryptography and Network Security, 5th International Conference, Zhuhai, China, June 5-8*, volume 4521 of *LNCS*, pages 18–30. Springer, 2007.

[14] Xiaofeng Chen, Fangguo Zhang, Willy Susilo, Haibo Tian, Jin Li, and Kwangjo Kim. Identity-based chameleon hash scheme without key exposure. In Ron Steinfeld and Philip Hawkes, editors, *Proceedings of ACISP, Information Security and Privacy - 15th Australasian Conference, Sydney, Australia, July 5-7*, volume 6168 of *LNCS*, pages 200–215. Springer, 2010.

[15] Xiaofeng Chen, Fangguo Zhang, Haibo Tian, Baodian Wei, Willy Susilo, Yi Mu, Hyunrok Lee, and Kwangjo Kim. Efficient generic on-line/off-line (threshold) signatures without key exposure. *Information Sciences*, 178(21):4192–4203, 2008.

[16] Dell and Intel. Manage your changing IT needs. A European Report on Servers & Storage for Small Business, February 2012. Available from `http://i.dell.com/sites/doccontent/business/smb/sb360/en/Documents/17595-Servers-Storage-Report-Feb-2012-PDF-V02-SM-LR-uk.pdf`, Accessed November 25, 2014.

[17] Yves Deswarte, Jean-Jacques Quisquater, and Ayda Saïdane. Remote integrity checking - how to trust files stored on untrusted servers. In Sushil Jajodia and Leon Strous, editors, *Proceedings of IFIP TC11/WG11.5, Integrity and Internal Control in Information Systems VI, Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS), Lausanne, Switzerland, November 13-14*, volume 140 of *IFIP*, pages 1–11. Springer, 2003.

[18] Premkumar T. Devanbu, Michael Gertz, Charles U. Martel, and Stuart G. Stubblebine. Authentic data publication over the internet. *Journal of Computer Security*, 11(3):291–314, 2003.

[19] C. Christopher Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *Proceedings of CCS, 16th ACM Conference on Computer and Communications Security, Chicago, Illinois, USA, November 9-13*, pages 213–222. ACM, 2009.

[20] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital schemes. In Gilles Brassard, editor, *Proceedings of CRYPTO, 9th Annual International Cryptology Conference*, volume 435 of *LNCS*, pages 263–275. Springer, 1989.

[21] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. Demonstrating data possession and uncheatable data transfer. *IACR Cryptology ePrint Archive*, 2006:150, 2006. Available from `http://eprint.iacr.org/2006/150`, Accessed November 25, 2014.

[22] FIPS. Digital Signature Standard (DSS). *National Institute for Standards and Technology*, 186-2:ii + 74, January 2000.

[23] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[24] Lein Harn. New digital signature scheme based on discrete logarithm. *Electronics Letters*, 30(5):396–398, March 1994.

[25] Lein Harn, Wen-Jung Hsin, and Changlu Lin. Efficient on-line/off-line signature schemes based on multiple-collision trapdoor hash families. *The Computer Journal*, 53(9):1478–1484, 2010.

[26] Alexander Heitzmann, Bernardo Palazzi, Charalampos Papamanthou, and Roberto Tamassia. Efficient integrity checking of untrusted network storage. In Yongdae Kim and William Yurcik, editors, *Proceedings of StorageSS, 4th ACM Workshop On Storage Security And Survivability, Alexandria, Virginia, USA, October 27-31*, pages 43–54. ACM, 2008.

[27] Patrick Horster, Markus Michels, and Holger Petersen. Meta-Multisignature schemes based on the discrete logarithm problem. In *Proceedings of IFIP/SEC: Eleventh International Conference on Information Security*, pages 128–141. Chapman and Hall, 1995.

[28] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of CCS, 14th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA, October 28-31*, pages 584–597. ACM, 2007.

[29] Vishal Kher and Yongdae Kim. Securing distributed storage: challenges, techniques, and systems. In Vijay Atluri, Pierangela Samarati, William Yurcik, Larry Brumbaugh, and Yuanyuan Zhou, editors, *Proceedings of the StorageSS, 1st International Workshop On Storage Security And Survivability, Fairfax, Virginia, USA, November 11*, pages 9–25. ACM, 2005.

[30] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Proceedings of NDSS, Network and Distributed System Security Symposium*. The Internet Society, 2000.

[31] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD, International Conference on Management of Data, Chicago, Illinois, USA, June 27-29*, pages 121–132. ACM, 2006.

[32] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Authenticated index structures for aggregation queries. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):32, 2010.

[33] Manish Mehta and Lein Harn. Efficient one-time proxy signatures. *IEE Proceedings Communications*, 152(2):129–133, 2005.

[34] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In *Proceedings of CCS, 8th ACM conference on Computer and Communications Security*, pages 245–254. ACM Press, 2001.

[35] Microsoft. Drivers & inhibitors to cloud adoption for small and midsize businesses. Findings of Microsoft SMB Business in the Cloud 2012 research report conducted in conjunction with Edge Strategies Inc., February 2012. Available from `http://www.microsoft.com/en-us/news/presskits/telecom/docs/SMBCloud.pdf`, Accessed December 4, 2012.

[36] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *ACM Transactions on Storage*, 2(2):107–138, 2006.

[37] Maithili Narasimha and Gene Tsudik. Authentication of outsourced databases using signature aggregation and chaining. In Mong-Li Lee, Kian-Lee Tan, and Vilas Wuwongse, editors, *Proceedings of DASFAA, 11th International Conference on Database Systems for Advanced Applications, Singapore, April 12-15*, volume 3882 of *LNCS*, pages 420–436. Springer, 2006.

[38] HweeHwa Pang, Jilian Zhang, and Kyriakos Mouratidis. Scalable verification for outsourced dynamic databases. *The Proceedings of the VLDB Endowment*, 2(1):802–813, 2009.

[39] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Proceedings of Advances in Cryptology - EUROCRYPT, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.

[40] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[41] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In Josef Pieprzyk, editor, *Proceedings of Advances in Cryptology - ASIACRYPT, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11*, volume 5350 of *LNCS*, pages 90–107. Springer, 2008.

[42] Mehul A. Shah, Ram Swaminathan, and Mary Baker. Privacy-preserving audit and extraction of digital contents. Cryptology ePrint Archive, Report 2008/186, 2008. Available from `http://eprint.iacr.org/`, Accessed November 25, 2014.

[43] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In *Proceedings of CRYPTO, 21st Annual International Cryptology Conference*, volume 2139 of *LNCS*, pages 355–367. Springer, 2001.

[44] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *Proceedings of CCS, 20th ACM SIGSAC Conference on Computer and Communications Security, Berlin, Germany, November 4-8*, pages 325–336. ACM, 2013.

[45] Jieping Wang and Xiaoyong Du. Skip list based authenticated data structure in DAS paradigm. In *Proceedings of GCC, 8th International Conference on Grid and Cooperative Computing, Lanzhou, Gansu, China, August 27-29*, pages 69–75. IEEE Computer Society, 2009.

[46] Weichao Wang, Zhiwei Li, Rodney Owens, and Bharat K. Bhargava. Secure and efficient access to outsourced data. In Radu Sion and Dawn Song, editors, *Proceedings of CCSW, the first ACM Cloud Computing Security Workshop, Chicago, Illinois, USA, November 13*, pages 55–66. ACM, 2009.

[47] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. Efficient query integrity for outsourced dynamic databases. In Ting Yu, Srdjan Capkun, and Seny Kamara, editors, *Proceedings of CCSW, 2012 ACM Workshop on Cloud computing security, Raleigh, North Carolina, USA, October 19*, pages 71–82. ACM, 2012.

[48] Yan Zhu, Gail-Joon Ahn, Hongxin Hu, Stephen S. Yau, Ho G. An, and Changjun Hu. Dynamic audit services for outsourced storages in clouds. *IEEE Transactions on Services Computing*, 6(2):227–238, 2013.