Privacy-Preserving Audit and Extraction of Digital Contents

Mehul A. Shah, Ram Swaminathan, Mary Baker
HP Laboratories, Palo Alto

HPL-2008-32R1
April 30, 2008*

auditing, storage, digital preservation, storage outsourcing, privacy, security

A growing number of online services, such as Google, Yahoo!, and Amazon, are starting to charge users for their storage. Customers often use these services to store valuable data such as email, family photos and videos, and disk backups. Today, a customer must entirely trust such external services to maintain the integrity of hosted data and return it intact. Unfortunately, no service is infallible.

To make storage services accountable for data loss, we present protocols that allow a third-party auditor to periodically verify the data stored by a service and assist in returning the data intact to the customer. Most importantly, our protocols are privacy-preserving, in that they never reveal the data contents to the auditor. Our solution removes the burden of verification from the customer, alleviates both the customer's and storage service's fear of data leakage, and provides a method for independent arbitration of data retention contracts.

Internal Accession Date Only Approved for External Publication

Privacy-Preserving Audit and Extraction of Digital Contents [*]

Mehul A. Shah Ram Swaminathan Mary Baker

HP Labs, {firstname.lastname}@hp.com

Abstract

A growing number of online services, such as Google, Yahoo!, and Amazon, are starting to charge users for their storage. Customers often use these services to store valuable data such as email, family photos and videos, and disk backups. Today, a customer must entirely trust such external services to maintain the integrity of hosted data and return it intact. Unfortunately, no service is infallible.

To make storage services accountable for data loss, we present protocols that allow a third-party auditor to periodically verify the data stored by a service and assist in returning the data intact to the customer. Most importantly, our protocols are privacy-preserving, in that they never reveal the data contents to the auditor. Our solution removes the burden of verification from the customer, alleviates both the customer's and storage service's fear of data leakage, and provides a method for independent arbitration of data retention contracts.

1 Introduction

A growing number of online services, such as Amazon, Yahoo!, Google, Snapfish, and Mozy.com, aim to profit by storing and maintaining lots of valuable user data. Example uses of this storage include online backup, email, photo sharing, and video hosting. Many of these service offer a small amount of "teaser" storage for free, and charge for larger, upgraded versions of the service.

Studies of deployed large-scale storage systems show that no storage service can be completely reliable; all have the potential to lose or corrupt customer data [4, 5, 19, 21]. Today, a customer that wants to rely on these services must make an uneducated choice. He has only negative newsworthy anecdotes on which to base his decision [13, 27], and service popularity or "brand name" is not a positive indicator of reliability [10]. To know if his data is safe, he must either blindly trust the service or laboriously retrieve the hosted data every time he wants to verify its integrity, neither of which is satisfactory. Unfortunately, to date, there are no fair and explicit mechanisms for making these services accountable for data loss.

Our proposed solution to provide storage service accountability is through independent, third-party auditing and arbitration. The customer and service enter into an agreement or contract for storing data in which the service provides some type of payment for data loss or failing to return the data intact, e.g. free prints, refunds, or insurance. In such an agreement, the two parties have conflicting incentives. The service provider, whose goal is to make a profit and maintain a reputation, has an incentive to hide data loss. On the other hand, customers are terribly unreliable, e.g. casual home users. Customers can innocently (but incorrectly) or fraudulently claim loss to get paid. Thus, we involve an independent, third party to arbitrate and confirm whether stored and retrieved data is intact.

[*] Also available as HP Labs Technical Report No. HPL-2008-32

Unfortunately, storage services are often reluctant to reveal their data to external parties for two reasons [3]. First, storage services have legal, e.g. HIPAA, and social incentives to maintain customers' privacy. Second, customer data and its derivatives have value, so services want to protect their business assets from competitors. Some efficient, challenge-response protocols have been proposed for auditing [1, 2, 11, 22, 23] storage services, but they all have the potential to reveal some information to the auditor. To address these concerns, we present third-party auditing and data retrieval protocols that never reveal the data contents to the third party.

1.1 Solution Overview

Our protocols have three important operations, *initialization*, *audit*, and *extraction*, and we primarily focus on the latter two. For audits, the auditor interacts with the service to check that the stored data is intact. For extraction, the auditor interacts with the service and customer to check that the data is intact and return it to the customer. These protocols have the following salient features:

- **Audit:** With minimal long-term state, an auditor can efficiently and repeatedly check stored contents on behalf of the customer. In these audits, the service must prove that contents are completely unchanged.

- **Extraction:** Upon retrieval, if the customer doubts the integrity of the data, the customer can use the extraction protocol which routes the data through the auditor to the customer. During extraction, the auditor can determine which party is at fault: whether the service lost data or which party is cheating by not obeying the protocol. Thus, the auditor can arbitrate a data retention contract.

- All our protocols do not reveal the data contents to the auditor. Our auditing protocols are zero-knowledge, providing no added information to the auditor. Our extraction protocols prevent an adversarial auditor from recovering the data contents. Yet, they still allow the auditor to check the integrity of retrieved data and forward it so that a customer can efficiently recover the contents.

- A customer does not need to maintain *any* long-term state. For example, he does not need to keep "fingerprints" or hashes to audit the stored data, or keep secret keys to decrypt the stored data upon retrieval.

A straightforward solution for maintaining privacy during audits is for the customer to encrypt his contents using symmetric-key encryption and keep those keys intact and secret from uninvited parties. Then, the auditor can use existing provably secure, challenge-response schemes [1, 2, 11, 23] on the encrypted contents. This solution is unsatisfactory because an unsophisticated customer is increasingly likely over time either to lose the keys and be unable to recover the contents, or to leak the keys [17].

In our protocols we shift the burden of keeping these secret keys to a storage service. Since services are already in the business of maintaining customers' data and privacy, the keys are safer with them. Keeping the data content private from the service is optional. A customer can keep the keys and encrypted data with the same service, thereby revealing the contents to that service and allowing it to provide value-added features beyond storage like search. Otherwise, the customer can separate the keys and encrypted data onto non-colluding services to maintain complete privacy.

The auditor is responsible for auditing and extracting *both* the encrypted data and the secret keys. Our protocols, however, never reveal the secret key to the auditor. Although we present the protocols for handling the encrypted data for completeness, they are straightforward extensions of existing techniques. Our main contributions are (1) in motivating and specifying the problem of privacy-preserving audit and extraction of digital data (2) privacy-preserving protocols for auditing and extracting the encryption key. In this paper, we present these protocols and show how they provably ensure data integrity and data privacy from the auditor.

Paper Organization. Section 2 lists the assumptions that motivate the need for both privacy-preserving auditing and extraction. Section 3 surveys other work in the same area and describes how our work is different from other recently proposed auditing and retrieval schemes. Section 4 describes various auditing and extraction protocols and proves that they meet the required properties with varying assumptions. Section 5 briefly discusses simple variations of our protocols and presents performance numbers and future work. Section 6 summarizes this work.

2 Basic Assumptions

We first describe the assumptions, incentives, and threats for ensuring integrity and privacy in our scenario.

Customers: Customers can be unsophisticated and terribly unreliable when it comes to preserving data. For example, they often accidentally or unknowingly lose data, e.g. by forgetting to back it up or by mistakenly overwriting or erasing files. Another common premise is that customers can maintain secret keys for the long term. We and others argue that this premise is invalid [6, 17, 26]. Customers are likely to lose, leak, or have their keys stolen over time. Diffie argues that the way "to strong security is less reliance on secrets" [9]. Thus, we assume that we cannot rely on customers to maintain any long-term state, such as secret keys or the hashes needed for auditing or extraction.

Although customers might not keep long-term secrets associated with the stored data, they still can use other methods to identify themselves to storage providers. For example, websites today often require users to authenticate themselves over the phone, mail, or through email to renew "short-term" passwords or identity tokens.

The following are two additional important assumptions about customers that shape our protocols. First, customers have an incentive to claim data loss fraudulently, for instance, to receive payment for losing data as specified by the retention contract. Second, customers want to keep their data private from third parties.

Storage Service: A storage service provides voluminous long-term storage. Such large-scale storage systems are complex and vulnerable to various threats that cause data loss. Baker et al. [6] list threats to data preservation. Some of these include: bugs in storage system software, e.g. drivers, filesystem, bugs in the hardware, e.g. block-storage firmware, bugs in the network path, and human error, e.g. malicious or accidental deletion. Thus, Shah et al. [24] argue that even well-run services must undergo end-to-end checks to verify the integrity of their stored data.

A storage service has an incentive to hide even the slightest data loss. News of data loss can tarnish service's reputation. Moreover, penalties in a data retention contract can diminish a service's profits. Thus, we assume a storage service may do whatever it can to fool an auditor into believing it has the original data. The service, however, does not have an exponential amount (in the size of the challenge during an audit) of storage or computing power. For reasons mentioned above, we assume the service has incentives against revealing stored data to anyone.

Stored data is useless unless the customer can retrieve and use it. On the other hand, a storage service, to protect its business, prefers customer lock-in. So, the service may be passive and fail to support extraction properly, hindering the customer from taking their data elsewhere. Or worse, the service may take steps to prevent data extraction altogether. In fact, Baker et al. [6] argue that lack of a reasonable “data exit” strategy is one of the main threats for long-term digital preservation. If a data retention contract specifies that extraction should be allowed, then hindering or preventing extraction must be caught. Our data extraction protocols, in which auditors assist in extraction, can catch or discourage such behavior.

Auditor: Using reliable, trusted, independent auditors is not unprecedented. The insurance industry and regulatory agencies have traditionally employed auditors when service providers and customers have conflicting goals [24], a practice from which we derive our setting and assumptions.

The auditor is independent, thus has no incentive to favor (or collude with) the service or customer in assigning blame. To allow contract arbitration, the storage service and customer must both trust the auditor in assigning blame. We assume the auditor, who is in the business of auditing, is more reliable than customers and, thus, can maintain a small amount of state for the long term. Finally, customer data and results derived from it have external value, so the auditor has a temptation to learn its contents.

Other: Our protocols rely upon computationally hard problems, in particular the discrete log. We assume that no party can efficiently (in polynomial time) solve this problem. Our protocol assumes external authentication methods exist and uses them for secure communication between all parties. Our protocols do not protect against denial of service attacks.

3 Related Work

Peer-to-peer storage systems: A few peer-to-peer (P2P) storage systems have been proposed for long-term data preservation. LOCKSS [18] is a P2P archival system for library periodicals. Although a library site in LOCKSS periodically performs audits of its content, there is no need for complete privacy since the contents are published. OceanStore [12] periodically checks stored data for integrity, but relies on users to keep secret keys for privacy and does not offer an interface for external auditing. Lillibridge *et al.* [14] present a P2P backup scheme in which peers request random blocks from their backup peers. This scheme can detect data loss from free-riding peers, but does not ensure *all* data is unchanged. Samsara [8] is a similar P2P system in which data owners query peers with replicas using a hash-based challenge-response protocol. In Samsara, however, the protocol only saves network bandwidth since it requires owners to maintain a copy of their data. Samsara nodes use probabilistic query and enforcement methods to dissuade peers from free-riding.

Auditing online storage services: Others have considered how to audit online storage services. Schwarz and Miller [22] describe a scheme in which an object is split across multiple data and parity chunks, and the service is asked to return algebraic signatures of various subsets of the chunks. This signature scheme is more efficient to compute than our cryptographic hashes, but does not simultaneously provide provable privacy and ensure full data integrity. Potshards is an archival system in which archives use this technique to query their peer archives and ensure data integrity [26]. Potshards keeps archived data private from unauthorized users (including other archives) through secret splitting, but is limited by the security of this auditing technique. In contrast, we are concerned with data privacy from a third-party auditor, and in our protocols, keeping data private from the service is orthogonal.

Lillibrigde and Eshghi describe two inventions that allow an archival service to commit provably to storing documents and customers to ensure provably that retrieved documents are indeed the original [15, 16]. The first scheme handles insertion into and retrieval from a collection without requiring the customer to keep long-term secrets, and the second handles document deletion. Unlike our protocols, these schemes require retrieving the entire document for auditing and reveal the committed contents to the auditor.

Efficient and Provably Strong Audit and Extraction Ateniese et al. [1] define the "provable data possession" (PDP) model for ensuring possession of file. They describe techniques based on homomorphic tags for auditing this file. Although their scheme is provably strong, their technique requires sufficient computation that it can be expensive for an entire file. They suggest randomly sampling a few blocks of the file. Although sampling is sufficient to catch gross omissions, small latent errors or data corruptions that perturb a few bits or single blocks may be missed. Such small errors are likely in storage systems [4, 5].

In a subsequent report, Ateniese et al. [2] describe a PDP scheme that uses symmetric-key encryption. It is similar to our scheme in that it relies on symmetric-key encryption and MACs to verify integrity of stored data. This method has lower-overhead than their previous scheme and still only provides probabilistic guarantees. This new scheme allows for updates, deletions, and appends to the stored file.

Juels and Kaliski [11] describe a "proof of retrievability" (POR) model for ensuring possession and retreivability of files. Their scheme embeds sentinels in a derivative of original file. The derivative is an encrypted version that has been redundantly coded for error correction. The redundancy prevents small errors and the auditing checks for sentinels that catches large omissions in the file. Moreover, extraction in their scheme is a by-product of auditing. Shacham and Waters [23] build on this and present a POR scheme that requires less communication.

All of these schemes are complementary to ours. They present methods for efficiently and provably ensuring that stored files are intact. Some ([1], [11], and [23]) allow a third-party audit the data. However, to keep customer data private from the auditor, the customer must encrypt the data and find a way to keep the decryption keys secret and intact for the long term. When used directly, these protocols are not provably privacy preserving and, thus, may leak bits of the customer data to the auditor. Moreover, none of these schemes considers privacy preserving extraction of customer data. For future work, we should consider adapting these related proofs of possession or retrievability models and auditing and extraction schemes to our setting.

4 Protocols

Our protocol has three phases: *initialization*, *audit*, and *extraction*. To simplify our discussion, we focus on storing, auditing, and retrieving a single customer object. For each of these phases, we describe how to handle both the encrypted data and encryption key.

Preliminaries: We assume all parties communicate through secure, reliable, authenticated channels for all phases. Using standard methods, we assume the customer, auditor, and service have agreed on a sufficiently large prime, p and a generator h of the group Z_p^*. To make the discrete log difficult, we chose p to be a safe prime — $p = 2q + 1$ where q is prime. With these parameters, the value $g = h^2 (\neq 1)$ generates all the quadratic residues of Z_p^*, a subgroup with order q. We will use operations in this subgroup (with g) for manipulating the encryption key. These values can be reused in all phases and for other instances of this protocol.

Protocol 4.1 Initialization

1. $\mathcal{C} \rightarrow \mathcal{S}$: K, $E_K(M)$, RC

1a. $\mathcal{C} \rightarrow \mathcal{A}$: W_c= g^K, X_c= $E_K(M)$, Y_c= RC

2. $\mathcal{S} \rightarrow \mathcal{A}$: W_s= g^K, X_s= $H(E_K(M))$, Y_s= RC

3. $\mathcal{A}$ checks $W_c = W_s$, $H(X_c) = X_s$, $Y_c = Y_s$

4.1 Initialization

The initialization phase ensures that the service and customer agree on the stored object and contract and prepares the auditor with state needed to audit. Initialization commits the service to storing the customer's object and binds the service and customer to a particular retention contract. In this phase, the auditor ensures both the service and customer agree on contents of the encrypted data and encryption key; otherwise, the auditor cannot resolve future disputes.

At start, the customer has an object M to archive and generates a secret key K, such that $1 < K < q$. We use $E_K(M)$ to denote symmetric-key encryption of M (e.g. using AES) with key K, $H(M)$ to denote a collision-free hash of M (e.g. using SHA-2). RC is the retention contract, which at least identifies the server and customer. We use $\mathcal{A}$ to denote the auditor, $\mathcal{C}$ to denote the customer, and $\mathcal{S}$ to denote the service. Initialization is described in protocol 4.1.

In step 1, the customer sends the information that the service must retain, both K and $E_K(M)$, along with the agreed-upon contract. In step 1a, the customer relays his understanding of the agreement with the service to the auditor. The customer sends a key-commitment, g^K, that fixes a value for the key without revealing the key, sends the encrypted data, which conceals the actual data, and sends the retention contract. In step 2, the service relays its understanding of the agreement with the customer to the auditor. This information includes the service's version of the key-commitment, g^K, a hash of the encrypted data, and retention contract.

At this point, $\mathcal{A}$ has enough information to check whether both $\mathcal{C}$ and $\mathcal{S}$ agree on a common key, the encrypted data, and the retention contract. He performs this check in step 3. If it passes, $\mathcal{A}$ sends *accept* to both $\mathcal{S}$ and $\mathcal{C}$ or sends *reject* to both $\mathcal{S}$ and $\mathcal{C}$.

Note, we must choose a sufficiently large K to prevent the auditor (or others) from breaking the encryption and $q > K$ in order for our scheme to hide and verify the key. We recommend AES for encryption, which supports 256 bit keys for encryption. Since K is small, the auditor's storage overhead for $g^K$is also small.

On accept, the auditor generates n random numbers $R_1, \ldots, R_n$ and computes l hashes $\widetilde{H}_1, \ldots, \widetilde{H}_n$. Each of these $\widetilde{H}_i$ are collision-free functions of both R_i and $E_K(M)$, and require knowledge of the *entirety* of $E_K(M)$. These hashes will be used in the auditing phase below to ensure the integrity of the encrypted data. One way to implement these hashes is to use HMACs, i.e. $\widetilde{H}_i = \text{HMAC}(R_i, E_K(M))$. We will use this definition for ease of exposition. To avoid the storage overhead of the encrypted data, the auditor discards it, and keeps the pairs: $L = \{(R_1,\ \widetilde{H}_1), \ldots, (R_l,\ \widetilde{H}_n)\}$.

In this protocol, we trust and rely on the auditor to maintain the tuple: $< g^K, H(E_K(M)), RC >$. The key-commitment will be used later to verify that the service still has the secret key, and the hash serves as a unique ID to reference the encrypted object.

4.2 Auditing

In the auditing phase, the auditor repeatedly checks the stored data using a challenge-response protocol. Each check establishes the data's integrity immediately before the check. For the remainder of the paper, we will show how to handle the encrypted data and encryption key, one after the other.

During auditing, the main threat from the storage service is that it lost some part of the encrypted data or encryption key and can fool the auditor into believing that it has both. It may fool the auditor in two main ways (1) by manipulating the current and cached past challenges the auditor provides or (2) by combining these challenges and derived values from the data or the encryption key, such that the encrypted data and encryption key cannot be entirely recovered from the derived values. Thus, for both the encrypted data and encryption key, we need to prove two properties (similar to standard proofs of knowledge) to ensure data integrity:

- **Completeness:** After receiving the challenge, if the service possesses all the bits of the encrypted data and encryption key, the auditor accepts the responses.
- **Soundness:** After receiving the challenge, if the service is missing any bit in the encrypted data or encryption key, the auditor accepts with negligible probability.

What entails "possession" of data bits ? Our definition is that the service can efficiently reproduce all bits of the encrypted data and encryption key. In this paper, efficient means polynomial in the size of the input (i.e. key or data).

The main threat from the auditor is that it may glean important information from the auditing process that could compromise the privacy guarantees provided by the service. For example, even a few bits from a file containing medical history could reveal whether a customer has a disease. To ensure privacy, we rely on different standards for the encrypted data and the encryption key. For the data, we rely on (1) the strength of the encryption scheme and (2) the zero-knowledge property of the protocol for encryption-key audits. Thus, we must prove the following for the encryption-key audits:

- **Zero-knowledge:** The service can be efficiently simulated such that the auditor's interaction is indistinguishable from one with a true service.

4.2.1 Encrypted Data Verification

We use a simple challenge-response protocol to check the encrypted data as described in protocol 4.2. The auditor's random challenge is selected and sent in steps (1-1a). The service's response is computed from this previously unseen challenge and sent in steps (2-2a), and the response is accepted or rejected in step (3). Although we could use any key-based collision resistant hash, our scheme is only practical when using a hash function whose range is small relative to the data size. This property limits the state that the auditor must remember. HMACs, in particular, have a range that is $O(1)$ in size, e.g. SHA-224 based HMACs are 224 bits long.

Theorem 1. *If L is non-empty (in step 1), the encrypted-data auditing protocol above is complete and sound.*

Proof. The completeness property is straightforward. Since the auditor generates the challenge-response pairs $< R_i, \widetilde{H}_i >$ from the encrypted data, if the service possesses the encrypted data, a correctly computed response will match.

Protocol 4.2 Encrypted Data Verification

1. $\mathcal{A}$ chooses any $R_j, \widetilde{H}_j$ from L and $L = L \setminus \{(R_j, \widetilde{H}_j)\}$.

1a. $\mathcal{A} \rightarrow \mathcal{S}$: R_j.

2. $\mathcal{S}$ computes $\widetilde{H}_s = HMAC(R_j, E_K(M))$.

2a. $\mathcal{S} \rightarrow \mathcal{A}$: $\widetilde{H}_s$.

3. $\mathcal{A}$ checks $\widetilde{H}_s = \widetilde{H}_j$ **else** declares $\mathcal{S}$ **lost** data.

Protocol 4.3 Regenerate set L, encrypted data challenge-response pairs.

1. $\mathcal{A} \rightarrow \mathcal{S}$: $X_a = H(E_K(M))$.

2. $\mathcal{S}$ retrieves $E_K(M)$ using X_a.

2a. $\mathcal{S} \rightarrow \mathcal{A}$: $E_K(M)$.

3. $\mathcal{A}$ verifies $X_a = H(E_K(M))$.

3a. $\mathcal{A}$ generates $(R_i, \widetilde{H}_i)$ pairs using $E_K(M)$.

To prove soundness, we must show that given the history of previous auditor interactions and precomputed values from previous possession, an adversarial service cannot fool the auditor into accepting. For this, we rely on the collision resistant properties of HMACs.

Assume for the purpose of contradiction that the auditor accepts a response, with non-negligible probability, from an adversarial service without all bits of $E_K(M)$. Let j denote the index of the current round. Then, there are four possibilities four step 2. (1) The service guessed the response. (2) The service precomputed $\widetilde{H}_j$ and replayed it. (3) The service replayed a response from previous round i such that HMAC$(R_i, E_K(M)) = \widetilde{H}_j$. (4) The service computed the response using a (potentially precomputed) value $V \neq E_K(M)$ such that HMAC$(R_j, V) = \widetilde{H}_j$.

All lead to contradictions. The first (1) occurs with negligible probability. The second (2) requires precomputing and storing an exponential number of responses, more than available storage. Finally, since $R_i \neq R_j$ with high probability (whp), both (3) and (4) contradict the collision resistance property of HMACs. Because the strength of HMACs depend on the underlying hash function, we recommend the SHA-2 family since they are still difficult to break. □

If the set L becomes empty, the auditor generates a new set by retrieving the data from the service using protocol 4.3. An adversarial service can only fool the auditor's check in 3a, if he is able to find a hash collision. Hence, this sub-protocol is also complete and sound.

4.2.2 Encryption Key Verification

To determine if the encryption key is in tact, we have a number of options. One option is to adapt existing identification schemes to prove the service has K without revealing K. For example, protocol 4.4 uses the Schnorr identification scheme [20] to show that the service still has K. Schnorr's scheme is complete and sound. For soundness, the service can fool the auditor into accepting with probability $\epsilon < 1/2^t$. But, this protocol is only provably zero-knowledge if the auditor honestly follows the protocol.

Protocol 4.4 Schnorr's identification scheme adapted to show encryption key, K, is intact.

1. $\mathcal{S}$ chooses random r s.t. $1 < r < q$.

1a. $\mathcal{S} \rightarrow \mathcal{A}$: $C_s = g^r$.

2. $\mathcal{A}$ chooses random z s.t. $1 < z < 2^t$, t is a security parameter.

2a. $\mathcal{A} \rightarrow \mathcal{S}$: z.

3. $\mathcal{S} \rightarrow \mathcal{A}$: $V_s = r - zK$.

4. $\mathcal{A}$ checks $(g^K)^z(g^{V_s}) = C_s$ **else** declares $\mathcal{S}$ **lost** key.

Protocol 4.5 Key Verification — Honest, but Curious Auditor Zero-Knowledge.

1. $\mathcal{A}$ chooses a random β s.t. $1 < \beta < q$ and computes g^β.

1a. $\mathcal{A} \rightarrow \mathcal{S}$: $V_a = g^\beta$.

2. $\mathcal{S}$ computes $W_s = (V_a)^K = g^{\beta K}$.

2a. $\mathcal{S} \rightarrow \mathcal{A}$: W_s.

3. $\mathcal{A}$ computes $W_a = (g^K)^\beta$

3a. $\mathcal{A}$ checks $W_a = W_s$ **else** declares $\mathcal{S}$ **lost** key.

We developed three additional protocols that rely on a non-standard assumption, the knowledge-of-exponent assumption, and provide different zero-knowledge guarantees. The first requires one round-trip, and, like Schnorr's scheme, is zero-knowledge for an auditor that is honest, i.e. follows the protocol. The second also takes one round-trip, but is zero-knowledge for all adversaries under the random oracle model. The third requires two round trips and also is zero-knowledge for all adversaries, but avoids the random oracle assumption.

Our schemes, reminiscent of Diffie-Hellman key-exchange, rely on the discrete log assumption (DLA) for privacy. To show soundness, however, we need to rely on the knowledge of exponent assumption (KEA) introduced by Damgard and later refined by others [7].

KEA 1. *For any adversary A that takes input (p, g, g^s) and returns (C, Y) such that $Y = C^s$, there exists an efficient "extractor" $\tilde{A}$, given the same inputs at A, returns x such that $g^x = C$.*

Informally, the original KEA assumption states that the only way to compute C^s is to know the value x. As per the previous work, an extractor is efficient if it takes polynomial time.

Protocol 4.5 shows our simplest key verification scheme that is zero-knowledge with an honest-but-curious auditor. In this protocol, the auditor computes the challenge in steps 1-1a, and the service responds in steps 2-2a. Because exponentiations commute, the auditor, in steps 3, can compute the expected answer directly from the key-commitment it has cached.

Theorem 2. *The first audit protocol 4.5 is complete and sound.*

Proof. Again, the completeness property is straightforward: Since exponentiation commutes, the auditor can generate the correct response from the key-commitment in step 3a.

Next, we show that the protocol is sound in the face of an adversarial service with a history of previous interactions and precomputed values from past possession of the key. Suppose an adversarial service generates a correct response in step 2, with non-negligible probability, without possessing K. Then, there are three possibilities for step 2. (1) The service guessed or used a precomputed value. (2) The service replayed a previous response. (3) The service has an algorithm to compute the result with inputs $(p, g, g^{\beta}, \{f(K)\})$, where $\{f(K)\}$ is a list of precomputed values from previous rounds such that $f()$ is one-way.

All three lead to contradictions. (1) Guessing occurs with negligible probability and precomputing the correct response is not possible because the service does not have exponential compute power or storage. (2) Since β is chosen at random from a large space, $\beta_i \neq \beta_j$, so $g^{\beta_i K} \neq g^{\beta_j K}$ for any previous round i with high probability.

(3) In this case, we show that having the additional {f(K)} does not help an adversarial service in computing the response without "knowing" K. In the initial round, the list $\{f(K)\}$ is empty. If the service successfully responds in the first round, then the service produced a result $Y = (g^K)^{\beta} = C^{\beta}$. Using KEA, there exists an efficient extractor that returns K.

Next, we show that if the service can fool the auditor in any subsequent round, then it could have tricked the auditor in the initial round (a contradiction). Suppose in round i, the service computes a correct response from the cached knowledge of $\{f(K)\}$ (computed in previous rounds), but cannot efficiently extract K. In that case, before the initial round, the service could simulate $i - 1$ rounds of interaction with an honest auditor, compute $\{f(K)\}$, and destroy K. Simulating $i - 1$ rounds is simple since the service only needs to generate random challenges g^{β}. Then, after the true challenge in the initial round, the service can compute the correct response with inputs $(p, g, g^{\beta}, \{f(K)\})$. This is a contradiction; hence, there exists an efficient extractor that returns K for all correct responses. Note, if poly-time extraction is not sufficiently efficient, then we need to strengthen our KEA assumption. □

Theorem 3. *The first audit protocol 4.5 is perfect zero-knowledge for an honest, but curious auditor that strictly follows the protocol.*

Proof. To show zero-knowledge, we show that an auditor can, by himself, simulate the transcript of an interaction with a correct, real service. A real interaction transcript contains $< g^{\beta_i}, g^{\beta_i K} >$ for each round i. A simulator uses g^K from initialization and β_i from the auditor's work tape to append $< g^{\beta_i}, (g^K)^{\beta_i} >$ for each round. The simulated transcript is identical to a true interaction. □

Although we believe the first protocol to be zero-knowledge with an arbitrarily malicious auditor, we have not yet produced a workable simulation. To prove zero-knowledge with such an adversary, we modify the protocol slightly. Instead of simply returning the response in step 2a, we hash the response from the service using a one-way, collision resistant hash, $H()$. Then, in the last step, compare the hashed values. That is, in step 2a, $W_s = H(g^{\beta K})$, and in step 3a, $W_a = H((g^K)^{\beta})$. Assuming that the hash function approximates a random function, this second protocol is zero-knowledge in the random oracle model.

Theorem 4. *The second audit protocol is complete and sound for all services, and zero-knowledge for all auditors under the random oracle model.*

The completeness and soundness proofs for this second protocol are straightforward because they follow the same line of reasoning as that for the first protocol. For soundness, we rely on the collision resistance property of the hash. The proof for zero-knowledge is slightly different:

Proof. We assume by DLA, that the auditor cannot recover K although he knows g^K. Since the auditor is no longer honest, he might not produce a value β that is consistent with the challenge

Protocol 4.6 Key Verification — Zero-Knowledge for all Auditors

1. $\mathcal{A}$ chooses a random β s.t. $1 < \beta < q$ and computes $V_a = g^{\beta}$.

1a. $\mathcal{A} \rightarrow \mathcal{S}$: V_a.

2. $\mathcal{S}$ checks V_a is a quadratic residue **else** $W_{s,1} = 0, W_{s,2} = 0$ goto **2b**.

2a. $\mathcal{S}$ chooses a random α s.t. $1 < \alpha < q$ and computes $W_{s,1} = (V_a)^{\alpha K} = g^{\beta\alpha K}$, $W_{s,2} = g^{\alpha K}$

2b. $\mathcal{S} \rightarrow \mathcal{A}$: $W_{s,1}$, $W_{s,2}$

3. $\mathcal{A}$ checks $(W_{s,2})^{\beta} = W_{s,1}$ **else** declares S **lost** key **end**.

3a. $\mathcal{A} \rightarrow \mathcal{S}$: β.

4. $\mathcal{S}$ checks $g^{\beta} = V_a$ **else** $\alpha = 1$.

4a. $\mathcal{S} \rightarrow \mathcal{A}$: α.

5. $\mathcal{A}$ checks $(g^K)^{\alpha} = W_{s,2}$ **else** declares S **lost** key.

in step 1. So, in this case, a true interaction of round i of the protocol contains the messages $< X_i, H(X_i^K) >$.

To show zero-knowledge, we simulate the service and simulate the hash function H as a random oracle. The simulation maintains a table with three columns: **B**, **C**, and **D**. Each row in this table records the query, c, to the random oracle, the simulated response, d, and the corresponding b, where $c = b^K$. The simulation for the service and the random oracle is as follows.

For every auditor challenge, X_i, the simulator looks it up the **B** column. If it exists, the simulator returns the corresponding d from that row as the service's response. Otherwise, the simulator considers all pairs (X_i, c) for all unpaired c in the table. The KEA assumption ensures that if the auditor previously computed a $c = X_i^K$, then there exists an extractor $\tilde{A}$ that, with input (p, g, g^K, X_i, c), returns β s.t. $g^{\beta} = X_i$. Feeding $\tilde{A}$ the (X_i, c) pairs, if $\tilde{A}$ returns a consistent β for some c in the table, the simulator fills that c's row with X_i for **B** and returns the corresponding d as the service's response. If no consistent β is found, the simulator creates a new row with X_i for **B**, selects a random R for **D**, adds the row to the table, and returns R as the service's response.

For every query, Y_j, to the random oracle by the auditor, the simulator performs an analogous process to ensure it returns consistent results. If Y_j exists in the **C** column, it returns the corresponding d from that row. Otherwise, the simulator queries $\tilde{A}$ with all pairs (b, Y_i) for all unpaired b in the table. Again, the KEA assumption ensures there is an efficient extractor $\tilde{A}$ that returns β s.t. $b = g^{\beta}$ if $Y_j = b^K$. If $\tilde{A}$ returns a consistent β s.t. $b = g^{\beta}$ and $Y_i = g^{\beta K}$, then the simulator fills that b's row with Y_i for **C** and returns the corresponding d value. Otherwise, the simulator adds a new row with Y_j for **C**, selects a random R for **D**, and returns R.

With such a simulator, all the queries to the random oracle (simulating H) will be consistent with previously returned simulated service responses and vice-versa. All the computations are polynomial in the input size. Most importantly, the auditor's transcript with the simulated service contains $< X_i, R >$, which is statistically indistinguishable from a true interaction. □

Protocol 4.6 shows our four-message key verification protocol that is computational zero-knowledge for all auditors. This protocol involves two round-trips. In the first steps (1-2b), the service obfuscates his response, with a random α so that the auditor cannot learn anything. The service also responds with a commitment indicating that he has αK. In the second round-trip, the service reveals α once he learns (in step 4) that the auditor has been honestly following the protocol. Thus, this protocol forces the auditor to follow the protocol.

Theorem 5. *The third audit protocol 4.6 is complete and sound for all services, and computational zero-knowledge for all auditors.*

Proof. **Completeness:** We show that an honest auditor will accept a correct service's responses. The auditor verifies that $W_{s,2} = g^{K\alpha}$ (step 5) and that $W_{s,2}^{\beta} = W_{s,1}$ (step 3). Thus, the auditor accepts (in step 5) iff $W_{s,1} = g^{\beta\alpha K}$, which is the value a correct service sends in step 2b.

Soundness: We rely on our KEA assumption for soundness. Suppose an adversarial service generates correct responses in steps 2 and 4, with non-negligible probability without possessing K. Let $\zeta = \alpha K$. Our argument works backward from step 5. At step 5, the auditor is sure that the services possesses α, g^K, and $W_{s,2} = g^{\zeta}$. If the service possesses ζ in step 3, then $K = \zeta\alpha^{-1} \pmod{q}$. Hence, the service can efficiently compute and, thus, possesses K, which is a contradiction.

Suppose the service does not possess ζ in step 3. At that step, the auditor is certain that the service knows $W_{s,1}$ and $W_{s,2}$ and that $W_{s,2}^{\beta} = W_{s,1}$. Since β is kept secret from the service, there are three possibilities for the responses at step 3. (1) The service guessed or used precomputed values. (2) The service replayed a previous response. (3) The service has an algorithm to compute the results with inputs $(p, g, g^{\beta}, \{f(\zeta)\})$, where $\{f(\zeta)\}$ is a list of precomputed values from previous rounds such that $f()$ is one-way.

All three lead to contradictions. (1) Guessing the right values occurs with negligible probability and precomputing responses is not possible because the service does not have exponential compute power or storage. (2) Since β is chosen at random from a large space, $\beta_i \neq \beta_j$, so $g^{\beta_i\zeta} \neq g^{\beta_j\zeta}$ for any previous round i with high probability. (3) The service produced a result $(C, Y) = (W_{s,2}, W_{s,1})$ s.t. $Y = C^{\beta}$. Using KEA, there exists an efficient extractor that returns ζ s.t. $g^{\zeta} = W_{s,2}$. As before, the additional $\{f(K)\}$ does not give the service any advantage in fooling the auditor.

Zero-Knowledge: To show zero-knowledge, we construct a simulator that generates an interaction transcript that is indistinguishable from an interaction with a real service. In this proof, the simulator has black-box access to an adversarial auditor.

For step 1, the simulator invokes the auditor for a challenge X_i and appends it to the transcript. For step 2, it chooses a random $1 < \alpha_i < q$, and appends $X_i^{\alpha_i}, g^{\alpha_i}$ to the transcript. Then, for step 3, it invokes the auditor and appends the returned β_i. If $g^{\beta_i} \neq X_i$, the simulator appends 1. Otherwise, if $g^{\beta_i} = X_i$, then the simulator rewinds the auditor to just before step 2. Instead, the simulator appends $(g^K)^{\alpha_i\beta_i}, (g^K)^{\alpha_i}, \beta_i, \alpha_i$ to the transcript.

For every round in which the auditor generates a valid β_i, the transcript contains the following: $< g^{\beta_i}, g^{\beta_i\alpha_i K}, g^{\alpha_i K}, \beta_i, \alpha_i >$. The simulated transcript is identical in a true interaction for this case. When the auditor generates an invalid β_x, the transcript contains $< X_i, X_i^{\alpha_i}, g^{\alpha_i}, \beta_x, 1 >$. A true interaction would have $< X_i, X_i^{\alpha_i K}, g^{\alpha_i K}, \beta_x, 1 >$. Since X_i is a quadratic residue, it has order q. Since $(K, q) = 1$, the distribution of $\alpha_i \pmod{q}$ and $\alpha_i K \pmod{q}$ are identical. Thus, the real and simulated interaction have the same distribution and the two transcripts are indistinguishable. □

4.3 Data Extraction

In the extraction phase, the auditor verifies that the data returned to the customer is the same as when it was ingested. Extraction starts with the service returning the encrypted data and key to the customer. Once received, the auditor uses one of the above verification protocols with the customer to ensure integrity. Typically, the data is intact and the customer does not contest the integrity of the data.

Protocol 4.7 Encrypted Data Extraction

```
 1.  $\mathcal{S} \rightarrow \mathcal{A}$ :    $E_K(M)$

 2.  $\mathcal{A}$ checks $H(E_K(M))$ with its cached copy **else** declares $\mathcal{S}$ **lost** data **end**.

2a.  $\mathcal{A} \rightarrow \mathcal{C}$ :    $E_K(M)$
```

If the check fails, the auditor is unsure whether the customer or the service is faulty. The service could have returned incorrect data or the customer could be buggy or maliciously trying to claim data loss. One option is to re-verify the stored data at the service and have the service resend the encrypted data and key to the customer. If after several tries verification continues to fail with the customer, we could simply conclude that the customer must be faulty. But, this policy is dangerous. The process for extracting data at the service may differ from the process of checking, so a simple bug could cause the service to be repeatedly faulty during extraction but not verification. Worse yet, since the service prefers customer lock-in, it may fail to fix the problem or actively hinder extraction.

To arbitrate such cases, we resort to a secondary protocol in which the auditor assists in returning the data to the customer. Henceforth, we simply refer to this secondary protocol as the extraction protocol. In extraction, the encrypted data and a "blinded" version of the encryption key are passed to the auditor. The auditor checks their integrity and forwards them to the customer. With these values, the customer can then recover the original data. We have several extraction protocols for the encryption key, each with different assumptions. In some of these, both the service or the customer may send bogus values that can prevent extraction from completing properly. Fortunately, these protocols allow the auditor to identify which party is sending the bogus values and therefore obstructing extraction.

For the extraction protocols, the analogues of the completeness, soundness, and privacy-preserving properties are as follows.

- **Completeness:** If the service returns the original copies, the auditor accepts and forwards the encrypted data to the customer and forwards enough information for the customer to efficiently compute the encryption key.

- **Soundness:** If the service is missing any bit of the encrypted data or encryption key, the auditor accepts the encrypted data or encryption key, respectively, with negligible probability.

  In some of our protocols, the auditor needs responses from the customer to determine if the service lost data. In such case, soundness includes the following. If a dishonest customer cheats to implicate the service, the auditor will reject the customer's responses with high probability.

- **Privacy-preserving:** We rely on the strength of the encryption scheme to protect all the bits of the encrypted data. Unfortunately, we do not have zero-knowledge proofs for most of our key-extraction protocols. That is, we could not construct an efficient simulator for the external parties (service and customer). Instead, for those protocols, we assume that if the key cannot be recovered in its entirety, the data is private from the auditor.

The procedure for extracting the encrypted data is straightforward, as shown in Protocol 4.7. Like data verification above, this protocol relies on the strength of the encryption scheme to ensure privacy of the data. This data extraction protocol is also complete and sound for all services. The

Protocol 4.8 Encrypted Key Extraction with Trusted Fourth Party

0. Assume that $\mathcal{S}$ and $\mathcal{C}$ agree on a random shared secret R s.t. $1 < R < q$ and $\mathcal{A}$ knows g^R which are established by a trusted fourth party.

1. $\mathcal{S} \rightarrow \mathcal{A}$: $\quad B_s = K + R \pmod{q}$.

2. $\mathcal{A}$ checks $g^{B_s} = g^K g^R = g^{K+R} \pmod{p}$ **else** declares $\mathcal{S}$ **lost** key (stop).

2a. $\mathcal{A} \rightarrow \mathcal{C}$: $\quad B_s$.

3. $\mathcal{C}$ computes $B_s - R = K \pmod{q}$.

proof is straightforward: completeness follows from the deterministic property of the hash function, and soundness follows from the collision resistance property of the hash.

4.3.1 Encryption Key Extraction

Protocol 4.8 is our simplest key-extraction protocol, which relies on a trusted fourth party and is zero-knowledge. The trusted external party generates and sends a shared secret R to the service and customer and sends a secret-commitment, g^R, to the auditor. The service sends a "blinded" version of the key in step 1, which appears no different from a random value. The auditor can check the blinded-key using the key-commitment and secret-commitment in step 2. In step 3, the customer recovers the original key.

Theorem 6. *Encryption key extraction protocol 4.8 is complete, sound, and zero-knowledge.*

Proof. Completeness is straightforward to check. We show soundness by contradiction. Assume the service does not know K. Suppose the auditor accepts in step 2 and the service sent a value $B_s = \alpha$. The check ensures that $\alpha = K + R$. Since the service knows R, the service can efficiently compute $K = \alpha - R$, contradiction.

We show zero-knowledge by simulating the other parties: service and trusted party. The simulator, acting as the trusted party, chooses a random β s.t. $1 < \beta < q$ and sends $g^{\beta}(g^K)^{-1} = g^{\beta - K}$ to the auditor. Acting as the service, the simulator sends β. The distribution of $< g^K, g^R, R + K \pmod{q} >$ is identical to $< g^K, g^{\beta-K}, \beta \pmod{q} >$. Thus, sending the blinded key with the secret commitment reveals no information to the auditor. □

Unfortunately, the above protocol involves an additional party for every extraction. Instead, since public-key infrastructure (PKI) already exists to manage identities and allow authenticated and private communication, we leverage this infrastructure for enabling extraction. Although PKI relies on trusted authorities, we involve them only for maintaining public-key certificates rather than for every extraction. We have several protocols that accomplish key extraction using public-private keys for encryption and signatures. To denote message M signed by $\mathcal{S}$, we write $M_{sig(\mathcal{S})}$.

Protocol 4.9 shows how to use RSA based encryption to transfer the key to the customer. In step 1, the service encrypts the key and sends a key-commitment and encryption to the auditor. In step 2, the auditor checks that the signature and key-commitment are consistent. Then, he forwards the pair to the customer. In step 3, the customer checks the signature and checks that the decrypted key corresponds to the key-commitment. If not, the customer sends the incorrect version of the key to the auditor. The signature ensures that the data comes from the service; otherwise, the auditor could inject a bogus value for A and cause the customer to reveal the true key. In step 4, the auditor checks that the encryption of the customer-returned key matches B, provided by the

Protocol 4.9 Encrypted Key Extraction using RSA

```
1.  $\mathcal{S}$ computes $D = \{A = g^K, B = K^e \pmod{N}\ \}$ where $(N, e)$ is $\mathcal{C}$ 's RSA public key.
1a. $\mathcal{S} \rightarrow \mathcal{A}:$    $D_{sig(\mathcal{S})}$.
2.  $\mathcal{A}$ checks $A = g^K$ and checks signature else declares $\mathcal{S}$ faulty (stop).
2a. $\mathcal{A} \rightarrow \mathcal{C}:$    $D_{sig(\mathcal{S})}$.
3.  $\mathcal{C}$ checks $\mathcal{S}$ 's signature else $F = \{Error, 0\}$ goto 3c.
3a. $\mathcal{C}$ decrypts $K' = B^d \pmod{N}$ where $d$ is $\mathcal{C}$ 's RSA private key.
3b. if $(g^{K'} = g^K)$: $F = \{OK\}$ else $F = \{Error, K'\}$.
3c. $\mathcal{C} \rightarrow \mathcal{A}:$    $F$.
4.  if $((F = \{Error, K'\})$ and $(g^K \neq g^{K'})$ and $(B = (K')^e \pmod{N}))$:
        $\mathcal{A}$ declares $\mathcal{S}$ faulty
    else
        $\mathcal{A}$ declares $\mathcal{C}$ faulty.
```

Protocol 4.10 Encrypted Key Extraction using Verifiable Encryption

```
1. $\mathcal{S} \rightarrow \mathcal{A}:$    $\hat{E}_{\mathcal{C}}(2K), \hat{W}$
2. $\mathcal{A}$ checks $V(\hat{E}_{\mathcal{C}}(2K), \hat{W}, g^K = h^{2K}, h) = 1$ else declares $\mathcal{S}$ lost key (stop).
3. $\mathcal{A} \rightarrow \mathcal{C}:$    $\hat{E}_{\mathcal{C}}(2K)$.
4. $\mathcal{C}$ decrypts $K = \hat{D}_{\mathcal{C}}(\hat{E}_{\mathcal{C}})2^{-1} \pmod{p}$.
```

service, and declares the service faulty, otherwise declares the customer faulty. Note, to protect his own privacy, we assume the customer never sends a correct value of the key to the auditor.

Protocol 4.9 is complete and sound. Completeness is straightforward. Soundness is also straightforward. If the service sends a bogus key commitment, the auditor will catch him in step 2, and if he sends a bogus key, the customer and auditor will catch him in steps 3 and 4. Moreover, if the customer sends an incorrect decryption of B, then the auditor will catch the customer in step 4. This latter holds because, with simple RSA encryption, the cipher-text of a message is the same on each invocation of encryption. Finally, we conjecture that the auditor cannot recover K using this protocol unless he can break RSA or the discrete-log.

Our next protocols are provably privacy-preserving and also use public-private key encryption. These protocols rely on the verifiable encryption scheme of Stadler [25]. Let $log_h(R) \pmod{p}$ denote the discrete logarithm of R base h, where h is a generator of Z_p^*. Moreover, let $\hat{E}_{\mathcal{C}}(M)$ denote the verifiable encryption of M using $\mathcal{C}$'s public key. Using Stadler's scheme, a party A can send an encrypted value of the discrete log of R, $\hat{E}_{\mathcal{C}}(log_h(R))$, to B along with a short witness, $\hat{W}$, that proves to B that the encryption is of the discrete log without revealing it. We denote this verification function, $V(\hat{E}, \hat{W}, R, h) \rightarrow \{0, 1\}$, which returns 1 on success. Stadler's scheme is based on ElGamal encryption [25].

Theorem 7. *Protocol 4.10 is complete, sound, and privacy-preserving.*

Protocol 4.10 shows how to use this scheme for key extraction. Since Stadler has shown that verification is zero-knowledge and the pair $\{h^{2K}, \hat{E}(2K)\}$ does not reveal the entirety of $2K$, our protocol is privacy-preserving [25]. Also, our protocol is complete and sound since Stadler's scheme is complete and sound. Note, there is no need for the service to sign the message it sends since the customer never reveals anything to the auditor.

In Stadler's scheme, the verification protocol requires the auditor to know the generator h (the square root of g) from initialization. Thus, the auditor must maintain it along with the other metadata. Moreover, the short proof, $\hat{W}$, relies on a cryptographic hash function such as SHA-256. Although practical, this proof's security depends on the hash function. We can modify Stadler's scheme in our setting to remove the use of such hash functions for key-extraction. This next extension is purely of theoretical interest since our protocols for the encrypted data also rely on such hash functions.

Protocol 4.11 is one such scheme that combines elements of the previous two extraction protocols 4.9 and 4.10. It splits Stadler's verification into two parts: part of the witness is provided by the service and the other by the customer. In step 1, the auditor encrypts $2K$ using standard ElGamal encryption (E) and computes one part of the witness, W.

In step 2, the auditor ensures that the witness values are consistent with the first element of the encryption, $E_1 = g^{\alpha}$. At this point, the auditor cannot verify that the second element of the encryption (E_2) contains the key since he does not have access to the customer's private key z. In fact, in the next step, the customer provides the necessary data to perform this check in case E_2 is invalid. However, to ensure that the customer is honest, the auditor, in step 2, chooses challenges that are indistinguishable from the first element of each pair in W. Then, the auditor forwards the key-commitment, the encryption, and $\{(x_{i,1}, x_{i,2})\}$, the challenges intermingled with the witness values from the service.

In step 3, the customer decrypts and checks that key is consistent with the key-commitment value. If not, he sends the second part of the witness, W', computed from the challenges and his private key, which the auditor will use to verify the encryption. In step 4, the auditor ensures that the customer correctly computed W'. If so, the auditor then checks to see if the encryption contains the discrete log of the key-commitment. The identity used for verification is essentially the same as that used by Stadler [25].

Theorem 8. *Protocol 4.11 is complete, sound, and privacy-preserving for a honest, but curious auditor.*

Proof. **Complete and Sound:** Completeness is straightforward to verify. For soundness, we need to show (1) if the service sends an incorrect encryption he will be caught (2) if the customer sends incorrect witness, he will be caught.

For (1), we assume the auditor and customer are correct. If the first element of the encryption is invalid, i.e. $E_1 \neq g^{\alpha}$, the auditor will discover that in step 2. The other possibility is that E_1 is consistent with the witness values, but E_2 is invalid. The service could have used an incorrect key or used the wrong multiplier for the encryption. In either case, the service will be caught in the second verification in step 4.

For (2), we assume the auditor and service are correct. In that case, the customer could send incorrect values in W' in step 3 to fool the auditor into rejecting the service's encryption in step 4. To compute W', the customer must exponentiate challenges $x_{i,1}$ and $x_{i,2}$ using his private key. For each i, in step 4, the auditor can verify that the customer gave the correct response for only one of $w'_{i,1}$ or $w'_{i,2}$; the one derived from ζ_i. The customer, however, cannot tell which challenge ($x_{i,1}$ or $x_{i,2}$) the service generated and which the auditor generated since they are drawn from the same distribution. So, for each i, the customer can fool the auditor with probability 1/2. Thus,

Protocol 4.11 Encrypted Key Extraction using Modified version of Stadler's Verifiable Encryption

```
0. Let $z$ s.t. $1 < z < q$ be $\mathcal{C}$ 's private key and $y = g^z$ be $\mathcal{C}$ 's public key.

1. $\mathcal{S}$ chooses a random $1 < \alpha < q$ and computes $E = (E_1 = g^\alpha, E_2 = (2K)^{-1} y^\alpha) \pmod p$.
   $\mathcal{S}$ chooses $k$ random $1 < \gamma_i < q$ and computes
      $W = \{(w_{i,1} = g^{\gamma_i} \pmod p),\ w_{i,2} = (\gamma_i - \alpha) \pmod q\ )\}$ for $1 \leq i \leq k$.
   $\mathcal{S} \rightarrow \mathcal{A}$ :    $(E, W)$.

2. $\mathcal{A}$ checks $(w_{i,1} \cdot g^{-w_{i,2}} = E_1)$ for $1 \leq i \leq k$ else declares $\mathcal{S}$ faulty (stop).
   for $1 \leq i \leq k$ do:
      $\mathcal{A}$ chooses random $1 < \zeta_i < q$.
      $\mathcal{A}$ chooses random bit, $b_i$.
      if $(b_i = 0)$:
         $x_{i,1} = w_{i,1}$ and $x_{i,2} = g^{\zeta_i} \pmod p$
      else
         $x_{i,1} = g^{\zeta_i} \pmod p$ and $x_{i,2} = w_{i,1}$
   $\mathcal{A} \rightarrow \mathcal{C}$ :    $(h, E, g^K, \{(x_{i,1}, x_{i,2})\}$ for $1 \leq i \leq k)$.

3. $\mathcal{C}$ decrypts: $M = (E_2 \cdot E_1^{-z})^{-1} = ((2K)^{-1} y^\alpha g^{-\alpha z})^{-1} = 2K \pmod p$.
   if $(h^M = g^K)$:                                    /* Key in cipher-text is correct. */
      $D = \{OK\}$
   else                                                 /* Key in cipher-text is incorrect. */
      $W' = \{(w'_{i,1} = h^{(x_{i,1})^z}, w'_{i,2} = h^{(x_{i,2})^z})\}$ for $1 \leq i \leq k$.
      $D = \{Error, W'\}$
   $\mathcal{C} \rightarrow \mathcal{A}$ :    $D$

4. $\mathcal{A}$ checks: if $(D = \{Error, W'\})$:
   /* Check if $\mathcal{C}$ 's response is valid */
   for $1 \leq i \leq k$ do:
      if $(b_i = 0)$:
         $v_{i,1} = w'_{i,1}$ and $v_{i,2} = w'_{i,2}$
      else
         $v_{i,1} = w'_{i,2}$ and $v_{i,2} = w'_{i,1}$          /* If $\mathcal{C}$ is honest, $v_{i,1} = h^{g^{\gamma_i z}}$ and $v_{i,2} = h^{g^{\zeta_i z}}$ */
      if $(v_{1,2} \neq h^{y^{\zeta_i}})$: $\mathcal{A}$ declares $\mathcal{C}$ faulty (stop).
   /* Check if $\mathcal{S}$ encrypted correctly */
   if $((g^K)^{(E_2 \cdot y^{w_{i,2}})} \neq v_{i,1}$ for all $i)$:       /* expands to $(g^K)^{((2K)^{-1} g^{\alpha z})(g^{z(\gamma_i - \alpha)})} \neq h^{g^{\gamma_i z}}$ */
      $\mathcal{A}$ declares $\mathcal{S}$ faulty.
   else
      $\mathcal{A}$ declares $\mathcal{C}$ faulty.
```

the overall probability that the auditor will fail to catch a dishonest customer is $1 - \frac{1}{2^k}$. This is negligible for sufficiently large k.

Privacy preserving: To show that the protocol is privacy preserving, we show that (1) the data that an honest service sends does not reveal the key and (2) the data that an honest customer sends in response to an auditor that follows the protocol (honest, but curious) does not reveal the key.

For (1), we note that this protocol is basically a variant of Stadler's scheme. Stadler showed that extracting $2K$ from $V = \{h^{2K}, g^{\alpha}, (2K)^{-1}g^{\alpha z}\}$ is as hard as the Decisional Diffie Hellman problem. Now, suppose the auditor could extract K with the additional information from the witness, W, using some algorithm, E, that takes input (V, W). Then, with E and just V alone, the auditor could extract the key, K. To do so, he picks k random β_i, computes $g^{\beta_i - \alpha}$, and uses these as the witness values ($w_{i,2}$ and $w_{i,1}$) along with V as the input to E. The simulated witness values have the same distribution as W from a true service; thus, W gives no additional knowledge. Since Stadler showed that recovering K from V is difficult, recovering K is still difficult with witness W.

For (2), the auditor receives responses for two types of challenges. The response for the first type results from the ζ_i that the auditor generates. That response, $h^{g^{\zeta_i z}}$, the auditor can generate himself (and does so in step 4). The response for the second type, $h^{g^{\gamma_i z}}$ is the value that the auditor computes from the encryption that the service provides. Thus, the customer offers no additional information and the auditor cannot recover either K or z. □

Note, unlike our second extraction protocol 4.9, protocol 4.11 never reveals K in plaintext to the auditor, even if the customer is faulty and incorrectly decrypts the data. Although we believe that protocol 4.11 is privacy-preserving with adversarial auditors that do not follow the protocol, we could not develop a workable simulation. A simple extension, therefore, is to use the same trick as we did in the auditing protocol 4.6. In step 3, the customer instead chooses a random $1 < \beta < q$ and returns $h^{(x_{i,1})^{\beta z}}$ and $h^{(x_{i,2})^{\beta z}}$ (which are $h^{g^{\gamma_i \beta z}}$ and $h^{g^{\zeta_i \beta z}}$). The auditor then sends the customer evidence, $\gamma_i - \alpha$ and ζ_i, that these challenges were formed according to the protocol. The customer verifies this evidence and then sends β to the auditor. Although this extension adds an additional round-trip, it makes the protocol provably zero-knowledge for all auditors. Thus, it is as secure as Stadler's interactive proof scheme without incurring numerous network round trips.

5 Discussion

5.1 Variations and Extensions

There are a number of variations of this protocol. We briefly mention a few.

- An important property of our protocols is that the minimal state needed for auditing, the key-commitment and the encrypted-data hash, reveals nothing about the stored data, so that state can be shared publicly. Thus, instead of storing that state with the auditor, we can store it with a public commitment storage service. Using such a service, we need not rely on a single auditor, but can use multiple auditors to audit our content. Thus, if auditors are faulty and Byzantine, we can use existing Byzantine agreement protocols to resolve disagreements.
- During initialization, instead of sending the data to both service and auditor, the customer can precompute the challenge-response pairs for the encrypted data and send the data to the service and the pairs to the auditor. However, we must worry about the customer sending fraudulent challenge-response pairs. In this case, the auditor can use the customer-supplied

challenges to audit and upon verification failure, he can retrieve the encrypted data from the service and check it against the non-keyed hash.

- For verification, instead of using our scheme based on cryptographic hashes for the encrypted data, we can use the provably strong techniques of Ateniese et al. [1], Shacham and Waters [23], or Juels and Kaliski [11]. Although these schemes might leak information when used on the plaintext, the first two are superior to our straightforward scheme in that they do not limit the number audits that can be performed. A simple hybrid of our scheme with one of these provides the privacy-preserving benefits of our scheme as well as the robustness of these other schemes.

5.2 Performance and Future Work

To support our protocols, storage services must export hooks for challenge-response queries and compute expensive functions for responses. To avoid these overheads, we can batch many files together into a single file and check that single file all at once. Since our protocols mostly send small hashes and auditing keys requires at most two exponentiations, the main overhead comes from computing HMACs over the entire contents. If we limit the number of checks, however, this overhead can be tolerable for a service.

We measured the performance of a SHA-1 HMAC over files stored on five 500GB SATA disks with a 2-core 2GHz Intel Xeon 5130 at 362 MB/s. At this peak rate, 50 CPUs in parallel can check 1PB in 16 hours. Spread over 30 days, this work imposes less than 2% overhead. Since many large-scale storage services handle an archival workload in which most data is rarely touched, checking monthly is reasonable.

For future work, we should consider modifying previous schemes [1, 2, 11, 23] to allow privacy-preserving audit and extraction. Certainly a hybrid as mentioned above is simple, but it still imposes the encryption and decryption overheads that the storage service and customer experience with our protocols. It may be possible to eliminate the encryption key altogether. To do so, we must first extend the formal definitions of proofs of possession or proofs of retrievability to include a notion of privacy from the auditor.

Finally, our current schemes require the auditor to be trusted and not collude with either party. In real world auditing situations in other contexts (e.g. financial auditing), such collusions can occur and have occurred in the past. In this case, a scheme that allows auditing by multiple auditors without the traditional overheads of Byzantine fault-tolerance techniques would be useful.

6 Conclusion

We describe methods for privacy preserving auditing and extraction of digital contents. Our two main contributions are (1) the motivation, assumptions, and setting for third-party privacy preserving auditing and extraction, and (2) various privacy-preserving protocols for auditing and extraction of data stored with a service provider. Auditing involves a third-party auditor that remotely verifies that the stored data are intact. For extraction, the auditor verifies the data is intact and returns it to the customer, ensuring that he received the original data. Thus, with our schemes, an auditor can arbitrate data retention contracts between storage provider and customer.

Our schemes divide the data into two pieces, an encryption key and the encrypted data. Our protocols allow an auditor, with minimal long-term state, to audit both those pieces and extract those pieces without revealing the underlying contents of either. Using our protocols, all these properties can be achieved without requiring the customer to maintain any long-term state (secret keys or hashes). The protocols for the encrypted data rely on cryptographic hashes and symmetric-key encryption. We present protocols for the encryption key that have varying assumptions, but all assume that the computing the discrete log is difficult. We believe protocols like ours will be necessary if outsourcing is truly to catch-on for preserving digital contents.

Acknowledgment

We thank Rob Schreiber and Alan Karp whose enthusiasm helped get this work started.

References

[1] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. Technical Report 2007/202, Cryptology ePrint Archive, December 2007.

[2] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudikd. Scalable and efficient provable data possession. Technical Report 2008/114, Cryptology ePrint Archive, March 2007.

[3] B. Parthasarathy, Snapfish Cofounder. Personal communication. 2005.

[4] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An analysis of latent sector errors in disk drives. In *SIGMETRICS'07*, pages 289–300, 2007.

[5] Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. An analysis of data corruption in the storage stack. In *USENIX conference on File and Storage Technologies (FAST '08)*, pages 223–238, 2008.

[6] Mary Baker, Mehul A. Shah, David S. H. Rosenthal, Mema Roussopoulos, Petros Maniatis, T. J. Giuli, and Prashanth Bungale. A fresh look at the reliability of long-term digital storage. In *EuroSys*, April 2006.

[7] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *CRYPTO '04*, 2004.

[8] Landon P. Cox and Brian D. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *SOSP*, October 2003.

[9] Whitfield Diffie. Perspective: Decrypting The Secret to Strong Security. `http://news.com.com/2010-1071-980462.html`, January 2003.

[10] Evan Hanson. Hotmail incinerates customer files. CNET News.com, `http://news.com.com/Hotmail+incinerates+customer+files/2100-1038_3-5226090.html`, June 2004.

[11] A. Juels and B. Kaliski. Pors: Proofs of retrievability for large files. In *ACM CCS*, pages 584–597, 2007.

[12] John Kubiatowicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage. *ASPLOS*, November 2000.

[13] David Lazarus. Precious Photos Disappear. San Francisco Chronicle, `http://www.sfgate.com/cgi-bin/article.cgi?file=/chronicle/archive/2005/02/02/BUG7QB3U0S1.DTL`, February 2005.

[14] Mark Lillibridge, Sameh Elnikety, Andrew Birrell, Mike Burrows, and Michael Isard. A cooperative internet backup scheme. *Usenix Annual Technical Conference*, pages 29–41, June 2003.

[15] Mark Lillibridge and Kave Eshghi. Method and system for assured document retention. *US Patent Filed.*, November 29, 2004. HP Doc #200401955.

[16] Mark Lillibridge and Kave Eshghi. Method and system for assured document retention with policy controlled deletion. *US Patent Filed.*, May 9, 2006. HP Doc #200404352.

[17] Petros Maniatis and Mary Baker. Enabling the Archival Storage of Signed Documents. In *FAST*, January 2002.

[18] Petros Maniatis, David S. H. Rosenthal, Mema Roussopoulos, Mary Baker, TJ Giuli, and Yanto Muliadi. Preserving Peer Replicas by Rate-Limited Sampled Voting. In *Proc. SOSP*, pages 44–59, Oct. 2003.

[19] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz A. Barroso. Failure trends in a large disk drive population. In *USENIX Conference on File and Storage Technologies (FAST 2007)*, pages 17–29, 2007.

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

[21] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *USENIX Conference on File and Storage Technologies (FAST 2007)*, pages 1–16, 2007.

[22] Thomas Schwarz and Ethan Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. *ICDCS*, July 2006.

[23] Hovav Shacham and Brent Waters. Compact proofs of retrievability. Technical Report 2008/073, Cryptology ePrint Archive, February 2008.

[24] Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, and Ram Swaminathan. Auditing to keep online storage services honest. In *HotOS XI*, May 2007.

[25] Markus Stadler. Publicly verifiable secret sharing. In *EUROCRYPT '96*, pages 190–199, 1996.

[26] Mark W. Storer, Kevin M. Greenan, and Ethan L. Miller. Potshards: Secure long-term storage without encryption. In *USENIX Annual Technical Conference*, June 2007.

[27] Alistair Wyse. PlusNet: Email Update - 3rd August. `http://usertools.plus.net/status/archive/1154603560.htm`, August 2006.