Lazy Revocation in Cryptographic File Systems

Michael Backes Christian Cachin
IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{mbc,cca}@zurich.ibm.com

Alina Oprea*
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
alina@cs.cmu.edu

Abstract

A crucial element of distributed cryptographic file systems are key management solutions that allow for flexible but secure data sharing. We consider efficient key management schemes for cryptographic file systems using lazy revocation. We give rigorous security definitions for three cryptographic schemes used in such systems, namely symmetric encryption, message-authentication codes and signature schemes. Additionally, we provide generic constructions for symmetric encryption and message-authentication codes with lazy revocation using key-updating schemes for lazy revocation, which have been introduced recently. We also give a construction of signature schemes with lazy revocation from identity-based signatures. Finally, we describe how our constructions improve the key rotation mechanism in the Plutus file system.

1. Introduction

Networked storage solutions, such as Network-Attached Storage (NAS) and Storage Area Networks (SAN), have emerged recently as an alternative to direct-attached storage. It is desirable that clients have similar security guarantees in these environments to those offered by traditional storage. However, the storage servers in a networked storage system are more exposed than direct-attached disks. Clients need to protect the confidentiality and integrity of the stored data themselves and can not rely on the storage servers for security guarantees. Cryptographic file systems have been designed for this task.

Sharing of information among clients is an important feature offered by file systems. Protecting data in non-cryptographic file systems relies on an access control mechanism, like the access control model of the Unix file system. Data sharing in cryptographic file systems is complicated by the problem of key management. While early cryptographic file systems did not address key management, recent systems offer diverse solutions. They range from fully centralized key distribution using a trusted key server [12] to completely decentralized key distribution done by the file system users [20, 19].

Access control granularity in a cryptographic file system affects the number of keys that need to be managed and the complexity of user revocation. Traditionally, access control is performed at the granularity of files and every file is protected by its own cryptographic keys. Another method, proposed in the Plutus file system [19], is to group files into *filegroups* with the same access control permissions and the same owner and to use the same cryptographic keys for all files in a filegroup. This method reduces the number of keys that need to be managed and distributed to users. In the rest of the paper, we assume that access control and key management are done for filegroups, but, nevertheless, our model can also be applied to the case in which keys are managed for each file individually.

Assuming that multiple users have access permissions for a filegroup, they need to share the keys of the filegroup. A *trusted entity*, which might either be a trusted key server or the owner of the filegroup, distributes the cryptographic keys for the filegroup. The users that have access rights to the filegroup might change over time. New users might be granted access to the filegroup, and existing users' access rights might be revoked. Initially, the same cryptographic keys can be used for all files in the filegroup, but once a revocation occurs, the keys need to be changed so that revoked users can not further perform cryptographic operations on files. It is thus necessary that the trusted entity changes the filegroup keys and distributes fresh keys to the users after every revocation. In addition, the cryptographic information computed with these keys (either ciphertext or integrity protection information for files) has to be recomputed.

There are two revocation models, depending on when

*Work done at IBM Research, Zurich Research Laboratory.

the cryptographic information is updated. In an *active revocation* model, all cryptographic information is immediately recomputed after a revocation takes place. This is expensive and might cause disruptions in the normal operation of the file system. In the alternative model of *lazy revocation*, the information for each file is recomputed only when the file is modified for the first time after a revocation [12]. Lazy revocation is more efficient than active revocation, and, in addition, revoked users do not get access to new information. But in systems with lazy revocation, key management becomes more difficult than in systems with active revocation because multiple keys might be used simultaneously for the files in the filegroup. These keys have to be stored and distributed to users upon request. Cryptosystems with efficient key management for file systems using lazy revocation are the focus of our work.

Contributions. This paper provides a comprehensive formalization of the cryptographic primitives used in a file system with lazy revocation. In our model, the cryptographic keys needed for operations on files are updated every time the trusted entity revokes a user. A user that has access rights to a filegroup receives from the trusted entity a *user key* that can be used to extract all keys needed for the cryptographic operations on the files. We define variations of symmetric encryption schemes, message-authentication codes and signature schemes with lazy revocation.

We give rigorous security definitions for the three cryptographic primitives. We also give generic constructions of symmetric encryption schemes and message-authentication codes with lazy revocation using the abstraction of key-updating schemes for lazy revocation, defined in a companion paper [2]. In addition, we give a generic transformation of identity-based signatures [27] to signature schemes with lazy revocation. Finally, we show how our primitives can be used in cryptographic file systems adopting lazy revocation.

Our lazy revocation model generalizes *key rotation*, a mechanism used previously for key management in the Plutus file system [19]. Using our constructions, we improve the key management scheme of the Plutus file system in two ways: first, the extraction of encryption keys for previous time intervals can be done more efficiently than key rotation in Plutus, using only symmetric-key operations, and, secondly, using signature schemes with lazy revocation, the storage space taken by the signature verification keys can be reduced from linear in the number of revocations to a constant.

Related work. Riedel et al. [26] survey the security of existing storage systems, in particular cryptographic file systems. Here we focus on key management schemes in these systems. The first cryptographic file systems (CFS [7, 8] and TCFS [10]) include simple key management schemes, not suitable for sharing large amounts of data. Cepheus [12] considers data sharing and uses a trusted key server for distributing cryptographic keys. Cepheus introduces the idea of lazy revocation, and implements it by storing all previous cryptographic keys for a filegroup on the trusted server.

Plutus [19] also adopts lazy revocation and introduces a sophisticated scheme for the derivation of previous cryptographic keys from the latest keys, called key rotation. Key rotation is applied to both the encryption keys and the signature keys for a filegroup. These keys are rotated forward by the owner applying the RSA permutation to the current key, using knowledge of the trapdoor information. Keys are rotated backward by users themselves using the public RSA permutation. Differentiation of readers and writers is done by distributing the file-signing key only to writers and the file-signature verification key only to readers.

The key rotation mechanism in the Plutus file system has been improved by several recent papers [13, 25, 2]. Fu et al. [13] formalize *key-regression*, a primitive equivalent to the model of key-updating schemes by Backes et al. [2]. Key-regression and key-updating schemes are cryptographic abstractions for key management in cryptographic file systems using lazy revocation. Fu et al. [13] also propose two more efficient constructions than the construction based on the RSA permutation from the Plutus file system. They use hash chains of symmetric-cryptographic primitives (either hash functions or block ciphers) for key updating. However, in these constructions the setup time is linear in the total number of revocations supported, as all the keys have to be generated in the initialization phase. Naor et al. [25] propose more efficient hash chains for the derivation of keys. By applying the fractal hash chain traversal method of Jakobsson [18], the time to update the cryptographic keys is decreased to at most a logarithmic number of hash computations at the expense of increasing the storage space to logarithmic in the total number of revocations. However, in both these proposals, the setup time and the time to extract previous keys are still linear. The binary-tree construction [2] achieves logarithmic cost for key updating and extraction of previous keys in the total number of revocations, logarithmic key storage in the number of revocations, and constant setup time.

In file systems such as Farsite [1], SNAD [24] and SiRiUS [14] the file data is protected by a unique file encryption key and/or a unique file signature key. The meta-data information for a file includes an encryption under the public key of each user with access rights to the file of these file keys. To perform a file operation, a user retrieves the encrypted meta-data information from the untrusted storage servers. While this scheme simplifies key management, it requires additional space on the storage servers proportional to the number of users accessing a file. To our knowledge, neither of these file systems addresses the problem of efficient revocation of users.

SUNDR [21] only provides data integrity, but not confidentiality. Every user signs files with its own signing key. A user checking the integrity of a file also needs to check that the user that signed the file still has write access to the file. SUNDR assumes a public-key infrastructure and a mechanism for distributing individual users' public keys to all users in the system.

2. Modeling Lazy Revocation

In systems adopting lazy revocation, the cryptographic keys used to perform operations on files need to be changed after every user revocation. We define a *time interval* to be the period between two user revocations. The total number of time intervals can be large. The trusted entity that is responsible for the cryptographic keys must change them at the beginning of each time interval and distribute the fresh keys to users having access to files.

Before providing the formal definition of our cryptographic primitives with lazy revocation, we recall the definition of *key-updating schemes for lazy revocation*, given in a companion paper [2]. Key-updating schemes for lazy revocation are an abstraction to manage the keys used for *symmetric* encryption and authentication algorithms for data storage systems with lazy revocation.

We do not consider here public-key encryption schemes with lazy revocation, as they do not have direct applications to storage systems. If needed in other applications, public-key encryption schemes with lazy revocations can be defined using our lazy revocation model. A construction similar to that of a forward-secure encryption scheme can be obtained from binary tree encryption schemes defined by Canetti, Halevi and Katz [9].

Key-updating schemes for lazy revocation. The model of key-updating schemes for lazy revocation consists of a trusted entity (called *center* in [2]) that manages the keys for a filegroup, and users that have access permissions to the filegroup. The trusted entity generates an initial state that is updated at the beginning of each time interval (corresponding to a revocation) and from which it can derive user keys upon request. A user can extract from a user key for a particular time interval the symmetric keys for all previous time intervals. We review the formal definition of key-updating schemes here.

Definition 1 (Key-Updating Schemes for Lazy Revocation [2]). A key-updating scheme consists of four deterministic polynomial-time algorithms $\mathsf{KU} = (\mathsf{Init}, \mathsf{Update}, \mathsf{Derive}, \mathsf{Extract})$ with the following properties:

– The initialization algorithm, Init, takes as input a *security parameter* 1^κ, a *number of time intervals* T, and a *random seed* s of length polynomial in κ and outputs an initial *trusted state* S_0.

– The key update algorithm, Update, takes as input the current *time interval* t, the current *trusted state* S_t, and outputs a *trusted state* S_{t+1} for the next time interval.

– The user key derivation algorithm, Derive, is given as input a *time interval* t, and the *trusted state* S_t, and outputs a *user key* M_t. The user key can be used to derive all keys k_i of previous time intervals, for $1 \leq i \leq t$.

– The key extraction algorithm, Extract, is executed by the user and takes as input a *time interval* t, the *user key* M_t for that time interval received from the trusted entity, and a *target time interval* $1 \leq i \leq t$. The algorithm outputs the *key* k_i for target time interval i.

We define the Init algorithm of a key-updating scheme to be deterministic because we can compose efficiently schemes with deterministic initialization algorithms. The *additive* and *multiplicative* composition methods [2] combine two key-updating schemes into a new scheme with the number of time interval either the sum or the product of the number of intervals of the two schemes. These methods are useful in building schemes with a large number of time intervals.

Security of key-updating schemes for lazy revocation. Informally, a key-updating scheme is secure if an adversary given the user keys for all consecutive time intervals up to some time t that is chosen adaptively, has no advantage in distinguishing the key for time interval $t+1$ from a randomly generated key. Formally, consider a probabilistic polynomial-time adversary $\mathscr{A}$ that participates in the following experiment:

Initialization: Given a random seed, the initial trusted state is generated with the Init algorithm.

Key compromise: The adversary adaptively picks a time interval t such that $0 \leq t < T$ as follows. Starting with $t = 0, 1, \ldots$, the adversary is given the user keys M_t for all consecutive time intervals until $\mathscr{A}$ decides to output stop or t becomes equal to $T-1$.

Challenge: A challenge for the adversary is generated, which is either the key for time interval $t+1$ generated with the algorithms of the key-updating scheme, or a random bit string of the appropriate length.

Guess: $\mathscr{A}$ outputs a bit b.

The key-updating scheme KU is secure if the advantage of the adversary of distinguishing between the key generated by KU for interval $t+1$ and the random key is only negligibly larger than $\frac{1}{2}$. For an adversary $\mathscr{A}$ and a scheme KU we denote $\mathsf{Adv}^{\mathsf{sku}}_{\mathsf{KU}}(\mathscr{A})$ its advantage. We denote $\mathsf{Adv}^{\mathsf{sku}}_{\mathsf{KU}}$ the maximum advantage of all adversaries.

Remark 1. Since we allow T to be exponential in the security parameter, we require that $\mathscr{A}$, a probabilistic polynomial-time algorithm, outputs stop at least once before halting. This requirement is placed on all cryptographic primitives for lazy revocation defined in this section, but is omitted in subsequent definitions for brevity.

Remark 2. This definition of security is equivalent to a definition in which the adversary can choose the challenge time interval t^* in which it has to distinguish between the keys, as long as $t^* > t$ and t^* is polynomial in the security parameter. We consider a game in which the adversary is challenged at time interval $t+1$ in all security definitions of cryptographic primitives for lazy revocation given in this paper.

Implementation. Three key-updating schemes are introduced in [2]: a *chaining construction* based on hash chains, a *trapdoor permutation* scheme derived from the key rotation method in Plutus [19], and a novel *tree construction*, which is the most efficient one among them.

3. Symmetric Encryption Schemes with Lazy Revocation (SE-LR)

In a cryptographic file system adopting lazy revocation, the file encryption keys must be updated by the trusted entity (e.g., the owner of the filegroup) as described above. Users might need to encrypt files using the encryption key of the current time interval or to decrypt files using *any* key of a previous time interval. Upon sending a corresponding request to the trusted entity, authorized users receive the *user key* of the current time interval from the trusted entity. Both the encryption and decryption algorithms take as input the user key, and the decryption algorithm additionally takes as input the index of the time interval for which decryption is performed.

3.1. Security Definitions

Before defining formally symmetric encryption schemes with lazy revocation, we first define symmetric encryption schemes and security against chosen-plaintext attacks (or *CPA-security*). We are interested in CPA-security as standard randomized modes of operation (e.g., cipher-block chaining) used with a block cipher modeled as a pseudorandom permutation satisfy this notion of security [4], but not stronger notions like security against chosen-ciphertext attacks.

Symmetric encryption schemes. A symmetric encryption scheme $\mathscr{E}$ consists of three algorithms: a key generation algorithm $\mathsf{Gen}(\cdot)$ that outputs a key (taking as input the security parameter), an encryption algorithm $\mathsf{Enc}_k(m)$ that outputs the encryption of a given message m with key k, and a decryption algorithm $\mathsf{Dec}_k(c)$ that decrypts a ciphertext c with key k. The first two algorithms might be probabilistic, but Dec is deterministic.

The correctness property requires that $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$, for all keys k generated by Gen and all messages m from the encryption domain.

CPA-security of a symmetric encryption scheme $\mathscr{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ requires that any polynomial-time adversary $\mathscr{A}$ with access to an encryption oracle $\mathsf{Enc}(\cdot)$ is unable to distinguish between encryption of two messages m_0 and m_1 of its choice. If $\mathscr{A}$ produces two messages whose encryptions it can distinguish with non-negligible probability, we say that $\mathscr{A}$ succeeds in breaking the CPA-security of scheme $\mathscr{E}$. We refer the reader to the paper by Bellare et al. [4] for formal definitions of CPA-security. For an adversary $\mathscr{A}$ and a symmetric encryption scheme $\mathscr{E}$ we denote $\mathsf{Adv}^{\mathsf{cpa}}_{\mathscr{E}}(\mathscr{A})$ its advantage. W.l.o.g., we can relate the success probability of $\mathscr{A}$ and its advantage as

$$\Pr\big[\mathscr{A} \text{ succeeds}\big] = \frac{1}{2}\big[1 + \mathsf{Adv}^{\mathsf{cpa}}_{\mathscr{E}}(\mathscr{A})\big]. \qquad (1)$$

Definition of SE-LR. Symmetric encryption schemes with lazy revocation include Init, Update and Derive algorithms for key generation that are similar to the corresponding algorithms of key-updating schemes, and secret-key encryption and decryption algorithms.

Definition 2 (Symmetric Encryption with Lazy Revocation)**.** A symmetric encryption scheme with lazy revocation consists of a tuple of five polynomial-time algorithms (Init, Update, Derive, Enc, Dec) with the following properties:

- The Init, Update and Derive deterministic algorithms have the same specification as the corresponding algorithms of a key-updating scheme.
- The probabilistic encryption algorithm, Enc, takes as input a *time interval* t, the *user key* M_t of the current time interval and a *message* m, and outputs a *ciphertext* c.
- The deterministic decryption algorithm, Dec, takes as input a *time interval* t, the *user key* M_t of the current time interval, the *time interval* i for which decryption is performed, and a *ciphertext* c, and outputs a *plaintext* m.

Correctness of $\mathsf{SE\text{-}LR}$. Suppose that $S_0 \leftarrow \mathsf{Init}(1^\kappa, T, s)$ is the initial trusted state computed from a random seed s, $S_i \leftarrow \mathsf{Update}(i, \mathsf{Update}(i-1, \ldots, \mathsf{Update}(0, S_0)\ldots))$ is the trusted state for time interval $i \leq T$ and $M_i \leftarrow \mathsf{Derive}(i, S_i)$ is the user key for time interval i. The correctness property requires that $\mathsf{Dec}(t, M_t, i, \mathsf{Enc}(i, M_i, m)) = m$, for all messages m from the encryption domain and all i, t with $i \leq t \leq T$.

CPA-security of $\mathsf{SE\text{-}LR}$. The definition of CPA-security for SE-LR schemes requires that any polynomial-time adversary with access to the user key for a time interval t that it may choose adaptively (and, thus, with knowledge of all keys for time intervals prior to t), and with access to an encryption oracle for time interval $t+1$ is not able to distinguish encryptions of two messages of its choice for time interval $t+1$.

Formally, consider a probabilistic polynomial-time adversary $\mathscr{A}$ that participates in the following experiment:

Initialization: Given a random seed, the initial trusted state S_0 is generated with the Init algorithm.

Key compromise: The adversary adaptively picks a time interval t such that $0 \leq t < T$ as follows. Starting with $t = 0, 1, \ldots$, the adversary is given the user keys M_t for all consecutive time intervals until $\mathscr{A}$ decides to output stop or t becomes equal to $T-1$.

Challenge: When $\mathscr{A}$ outputs stop, it also outputs two messages, m_0 and m_1. A random bit b is selected and $\mathscr{A}$ is given a challenge $c = \mathsf{Enc}(t+1, M_{t+1}, m_b)$, where M_{t+1} is the user key for time interval $t+1$ generated with the Init, Update and Derive algorithms.

Guess: $\mathscr{A}$ has access to an encryption oracle $\mathsf{Enc}(t+1, M_{t+1}, \cdot)$ for time interval $t+1$. At the end of this phase, $\mathscr{A}$ outputs a bit b' and succeeds if $b = b'$.

The SE-LR scheme is CPA-secure if the adversary succeeds in this game with probability only negligibly larger than $\frac{1}{2}$. For an adversary $\mathscr{A}$ and a SE-LR scheme $\mathscr{E}^{\mathtt{lr}}$ we denote $\mathsf{Adv}^{\mathtt{cpa\text{-}lr}}_{\mathscr{E}^{\mathtt{lr}}}(\mathscr{A})$ its advantage. W.l.o.g., we can relate the success probability of $\mathscr{A}$ and its advantage as

$$\Pr\big[\mathscr{A} \text{ succeeds}\big] = \frac{1}{2}\big[1 + \mathsf{Adv}^{\mathtt{cpa\text{-}lr}}_{\mathscr{E}^{\mathtt{lr}}}(\mathscr{A})\big]. \qquad (2)$$

Tweakable ciphers. A tweakable block cipher [22, 17] is similar to a symmetric encryption scheme with the difference that it is deterministic and both the encryption and decryption algorithms take an additional parameter, called *tweak*. Such ciphers must be length-preserving and require that encryptions are indistinguishable as long as they are produced with different tweaks. We do not define tweakable ciphers here, but the interested reader can consult [17] for formal definitions. Tweakable ciphers with lazy revocation can be defined and implemented in a similar way as symmetric encryption schemes with lazy revocation. We omit here the details.

3.2. Generic Construction

Let KU = (Init, Update, Derive, Extract) be a secure key-updating scheme and $\mathscr{E}$ = (Gen, Enc, Dec) a CPA-secure symmetric encryption scheme such that the keys generated by KU have the same length as those generated by $\mathscr{E}$. We construct a symmetric encryption scheme with lazy revocation $\mathscr{E}^{\mathtt{lr}} = (\mathsf{Init}^{\mathtt{lr}}, \mathsf{Update}^{\mathtt{lr}}, \mathsf{Derive}^{\mathtt{lr}}, \mathsf{Enc}^{\mathtt{lr}}, \mathsf{Dec}^{\mathtt{lr}})$ as follows:

- The $\mathsf{Init}^{\mathtt{lr}}$, $\mathsf{Update}^{\mathtt{lr}}$, and $\mathsf{Derive}^{\mathtt{lr}}$ algorithms of $\mathscr{E}^{\mathtt{lr}}$ are the same as the corresponding algorithms of KU.
- The $\mathsf{Enc}^{\mathtt{lr}}(t, M_t, m)$ algorithm runs $k_t \leftarrow \mathsf{Extract}(t, M_t, t)$ and outputs $c \leftarrow \mathsf{Enc}_{k_t}(m)$.
- The $\mathsf{Dec}^{\mathtt{lr}}(t, M_t, i, m)$ algorithm runs $k_i \leftarrow \mathsf{Extract}(t, M_t, i)$ and outputs $m \leftarrow \mathsf{Dec}_{k_i}(c)$.

Theorem 1. *Suppose that* KU *is a secure key-updating scheme for lazy revocation and* $\mathscr{E}$ *is a CPA-secure symmetric encryption scheme. Then* $\mathscr{E}^{\mathtt{lr}}$ *is a CPA-secure symmetric encryption scheme with lazy revocation.*

Proof. Correctness is easy to see. To prove CPA-security of $\mathscr{E}^{\mathtt{lr}}$, let $\mathscr{A}^{\mathtt{lr}}$ be a polynomial-time adversary algorithm successful in breaking the CPA-security of $\mathscr{E}^{\mathtt{lr}}$. We construct an adversary $\mathscr{A}$ that breaks the CPA-security of $\mathscr{E}$:

- $\mathscr{A}$ is given access to an encryption oracle $\mathsf{Enc}(\cdot)$.
- $\mathscr{A}$ generates a random seed s and uses this to generate an instance of the scheme KU.
- $\mathscr{A}$ gives to $\mathscr{A}^{\mathtt{lr}}$ the user keys M_t from the instance of scheme KU generated in the step above.
- When $\mathscr{A}^{\mathtt{lr}}$ outputs stop at time interval t and two messages, m_0 and m_1, $\mathscr{A}$ also outputs m_0 and m_1.
- $\mathscr{A}$ is given challenge c and it gives this challenge to $\mathscr{A}^{\mathtt{lr}}$.
- When $\mathscr{A}^{\mathtt{lr}}$ makes a query to the encryption oracle for time interval $t+1$, $\mathscr{A}$ replies to this query using the encryption oracle $\mathsf{Enc}(\cdot)$.
- $\mathscr{A}$ outputs the same bit as $\mathscr{A}^{\mathtt{lr}}$.

From the construction of the simulation it follows that

$$\Pr\big[\mathscr{A} \;\text{ succeeds}\big] = \Pr[\mathscr{A}^{\mathtt{lr}} \text{ succeeds } |E\big],$$

where E is the event that $\mathscr{A}^{\mathtt{lr}}$ does not distinguish the simulation done by $\mathscr{A}$ from the CPA game defined in Section 3. The only difference between the simulation and the CPA game is that $\mathscr{A}$ uses in the simulation the encryption oracle with a randomly generated key to reply to encryption queries for time interval $t+1$, whereas in the CPA game the encryption is done with key k_{t+1} generated with the Update, Derive and Extract algorithms of scheme KU. By the definition of E, we have $\Pr[\bar{E}] \leq \mathsf{Adv}^{\mathtt{sku}}_{\mathsf{KU}}$.

We can bound the probability of success of $\mathscr{A}^{\mathtt{lr}}$ as:

$$\begin{aligned} \Pr\big[\mathscr{A}^{\mathtt{lr}} \text{ succeeds}\big] &= \Pr\big[\mathscr{A}^{\mathtt{lr}} \text{ succeeds } |E\big]\Pr\big[E\big] + \\ &\quad \Pr\big[\mathscr{A}^{\mathtt{lr}} \text{ succeeds } |\bar{E}\big]\Pr\big[\bar{E}\big] \\ &\leq \Pr\big[\mathscr{A}^{\mathtt{lr}} \text{ succeeds } |E\big] + \Pr\big[\bar{E}\big] \\ &\leq \Pr\big[\mathscr{A} \text{ succeeds }\big] + \mathsf{Adv}^{\mathtt{sku}}_{\mathsf{KU}}. \quad (3) \end{aligned}$$

Using (1), (2), and (3) we obtain

$$\mathsf{Adv}^{\mathtt{cpa\text{-}lr}}_{\mathscr{E}^{\mathtt{lr}}}(\mathscr{A}^{\mathtt{lr}}) \leq \mathsf{Adv}^{\mathtt{cpa}}_{\mathscr{E}}(\mathscr{A}) + 2\mathsf{Adv}^{\mathtt{sku}}_{\mathsf{KU}}.$$

The CPA-security of $\mathscr{E}$ and the security of the key-updating scheme KU imply that $\mathsf{Adv}^{\mathtt{cpa}}_{\mathscr{E}}(\mathscr{A})$ and $\mathsf{Adv}^{\mathtt{sku}}_{\mathsf{KU}}$ are negligible. It follows that $\mathsf{Adv}^{\mathtt{cpa\text{-}lr}}_{\mathscr{E}^{\mathtt{lr}}}(\mathscr{A}^{\mathtt{lr}})$ is negligible, which proves the statement of the theorem. □

Implementation. In practice, we can instantiate the CPA-secure symmetric-encryption scheme with a block cipher (such as AES) in one of the CPA-secure modes of operation [23] (e.g., cipher-block chaining). The most efficient key-updating scheme is our binary tree construction proposed in [2], which only performs symmetric-key operations (more specifically, pseudo-random function applications implemented again by a block cipher). Its Update, Derive and Extract algorithms have logarithmic complexity and its trusted state and user key sizes are logarithmic in the total number of time intervals.

Suppose that AES with 128-bit key size is used for the derivation of the keys. In a system that supports up to 1000 revocations, at most 10 AES computations need to be done for the Update, Derive and Extract algorithms. The center state and user keys consist of up to 10 AES keys or 160 bytes each. This adds a very small overhead to the cost of file data encryption. Details of the binary-tree construction are given in a companion paper [2].

4. Message-Authentication Codes with Lazy Revocation (MAC-LR)

If message-authentication codes are used for providing integrity in a cryptographic file system, then a secret key for computing and verifying authentication tags needs to be distributed to all authorized users. The users generate a tag using the key of the current time interval and may verify tags for any of the previous time intervals with the corresponding keys. Similar to symmetric-key encryption with lazy revocation, both the tagging and verification algorithms need to take as input the current user key, and the verification algorithm additionally takes as input the index of the time interval at which the tag was generated.

4.1. Security Definitions

Before defining message-authentication codes with lazy revocation, we recall the definitions of message authentication codes and their security under chosen-message attacks (or *CMA-security*).

Message-authentication codes. A message-authentication code (MAC) consists of three algorithms: a key generation algorithm $\mathsf{Gen}(\cdot)$ that outputs a key (taking as input a security parameter κ), a tagging algorithm $\mathsf{Tag}_k(m)$ that outputs the authentication tag τ of a given message m with key k, and a verification algorithm $\mathsf{Ver}_k(m,\tau)$ that outputs a bit. A tag τ is said to be *valid* on a message m for a key k if $\mathsf{Ver}_k(m,\tau) = 1$. The first two algorithms might be probabilistic, but Ver is deterministic.

The correctness property requires that $\mathsf{Ver}_k(m, \mathsf{Tag}_k(m)) = 1$, for all keys k generated with the Gen algorithm and all messages m from the message space.

CMA-security for a message-authentication code [5] requires that any polynomial-time adversary with access to a tagging oracle $\mathsf{Tag}(\cdot)$ is not able to generate a message and a valid tag for which it did not query the tagging oracle.

Definition of MAC-LR. Message-authentication codes with lazy revocation include Init, Update and Derive algorithms for key generation that are similar to the corresponding algorithms of key-updating schemes, and secret-key tagging and verification algorithms.

Definition 3 (Message-Authentication Codes with Lazy Revocation)**.** A message-authentication code with lazy revocation consists of a tuple of five polynomial-time algorithms (Init, Update, Derive, Tag, Ver) with the following properties:

- The Init, Update and Derive deterministic algorithms have the same specification as the corresponding algorithms of a key-updating scheme.
- The probabilistic tagging algorithm, Tag, takes as input a *time interval* t, the *user key* M_t of the current time interval and a *message* m, and outputs an authentication *tag* τ.
- The deterministic verification algorithm, Ver, takes as input a *time interval* t, the *user key* M_t of the current time interval, the *time interval* i for which verification is performed, a *message* m, and a *tag* τ, and outputs a

bit. A tag τ computed at time interval i is said to be *valid* on message m if $\mathsf{Ver}(t, M_t, i, m, \mathsf{Tag}(i, M_i, m)) = 1$ for some $t \geq i$.

Correctness of MAC-LR**.** Suppose that $S_0 \leftarrow \mathsf{Init}(1^\kappa, T, s)$ is the initial trusted state computed from a random seed s, $S_i \leftarrow \mathsf{Update}(i, \mathsf{Update}(i-1, \ldots, \mathsf{Update}(0, S_0)\ldots))$ is the trusted state for time interval $i \leq T$ and $M_i \leftarrow \mathsf{Derive}(i, S_i)$ is the user key for time interval i. The correctness property requires that $\mathsf{Ver}(t, M_t, i, m, \mathsf{Tag}(i, M_i, m)) = 1$, for all messages m from the message space and all i, t with $i \leq t \leq T$.

CMA-security of MAC-LR**.** The definition of security for MAC-LR schemes requires that any polynomial-time adversary with access to the user key for a time interval t that it may choose adaptively (and, thus, with knowledge of all keys for time intervals prior to t), and with access to a tagging oracle for time interval $t+1$ is not able to create a valid tag on a message not queried to the tagging oracle.

Formally, consider a probabilistic polynomial-time adversary $\mathscr{A}$ that participates in the following experiment:

Initialization: Given a random seed, the initial trusted state S_0 is generated with the Init algorithm.

Key compromise: The adversary adaptively picks a time interval t such that $0 \leq t < T$ as follows. Starting with $t = 0, 1, \ldots$, the adversary is given the user keys M_t for all consecutive time intervals until $\mathscr{A}$ decides to output stop or t becomes equal to $T-1$.

Tag generation: $\mathscr{A}$ has access to a tagging oracle $\mathsf{Tag}(t+1, M_{t+1}, \cdot)$ for time interval $t+1$ and outputs a message m and a tag τ.

The adversary is successful in breaking the CMA-security of the MAC if m was not a query to the tagging oracle and τ is a valid tag on m for interval $t+1$. The MAC-LR scheme is CMA-secure if the adversary succeeds in this game only with negligible probability.

4.2. Generic Construction

Let KU = (Init, Update, Derive, Extract) be a secure key-updating scheme and MA = (Gen, Tag, Ver) a CMA-secure message-authentication code such that the keys generated by KU have the same length as those generated by MA. We construct a message-authentication code with lazy revocation $\mathsf{MA}^{\mathtt{lr}} = (\mathsf{Init}^{\mathtt{lr}}, \mathsf{Update}^{\mathtt{lr}}, \mathsf{Derive}^{\mathtt{lr}}, \mathsf{Tag}^{\mathtt{lr}}, \mathsf{Ver}^{\mathtt{lr}})$ as follows:

– The $\mathsf{Init}^{\mathtt{lr}}$, $\mathsf{Update}^{\mathtt{lr}}$, and $\mathsf{Derive}^{\mathtt{lr}}$ algorithms of scheme $\mathsf{MA}^{\mathtt{lr}}$ are the same as the corresponding algorithms of KU.

– The $\mathsf{Tag}^{\mathtt{lr}}(t, M_t, m)$ algorithm runs $k_t \leftarrow \mathsf{Extract}(t, M_t, t)$ and outputs $c \leftarrow \mathsf{Tag}_{k_t}(m)$.

– The $\mathsf{Ver}^{\mathtt{lr}}(t, M_t, i, m, \tau)$ algorithm runs $k_i \leftarrow \mathsf{Extract}(t, M_t, i)$ and outputs the value returned by $\mathsf{Ver}_{k_i}(m, \tau)$.

Theorem 2. *Suppose that* KU *is a secure key-updating scheme for lazy revocation and* MA *is a CMA-secure message-authentication code. Then* $\mathsf{MA}^{\mathtt{lr}}$ *is a secure message-authentication code with lazy revocation.*

Proof. Correctness is easy to see. To prove CMA-security for $\mathsf{MA}^{\mathtt{lr}}$, let $\mathscr{A}^{\mathtt{lr}}$ be a polynomial-time adversary algorithm successfully in breaking the security of $\mathsf{MA}^{\mathtt{lr}}$. We construct an adversary $\mathscr{A}$ that breaks the security of MA:

– $\mathscr{A}$ is given access to a tagging oracle $\mathsf{Tag}(\cdot)$.

– $\mathscr{A}$ generates a random seed s and uses this to generate an instance of scheme KU.

– $\mathscr{A}$ gives to $\mathscr{A}^{\mathtt{lr}}$ the user keys M_t from the instance of scheme KU generated in the step above.

– When $\mathscr{A}^{\mathtt{lr}}$ makes a query to the tagging oracle for time interval $t+1$, $\mathscr{A}$ replies to this query using the tagging oracle $\mathsf{Tag}(\cdot)$.

– $\mathscr{A}$ outputs the same message and tag pair as $\mathscr{A}^{\mathtt{lr}}$.

From the construction of the simulation it follows that

$$\Pr\big[\mathscr{A} \text{ succeeds}\big] = \Pr[\mathscr{A}^{\mathtt{lr}} \text{ succeeds } | E\big],$$

where E is the event that $\mathscr{A}^{\mathtt{lr}}$ does not distinguish between the simulation done by $\mathscr{A}$ and the MAC game from Section 4. Using a similar argument as in the proof of Theorem 1, we can bound $\Pr[\bar{E}] \leq \mathsf{Adv}^{\mathtt{sku}}_{\mathsf{KU}}$. It is immediate, as in the proof of Theorem 1 that

$$\Pr\big[\mathscr{A}^{\mathtt{lr}} \text{ succeeds }\big] \leq \Pr\big[\mathscr{A} \text{ succeeds }\big] + \mathsf{Adv}^{\mathtt{sku}}_{\mathsf{KU}},$$

and the security of schemes KU and MA implies the conclusion of the theorem. □

Implementation. In practice, there are many efficient MAC schemes, such as CBC-MAC [23] or HMAC [3]. They can be combined with key-updating schemes and achieve the same complexities as the implementation of symmetric encryption schemes with lazy revocation.

5. Signature Schemes with Lazy Revocation (SS-LR)

Signature schemes can be used for providing integrity of files. When differentiation of readers and writers is desired, a MAC is not sufficient because it is a symmetric primitive, and an asymmetric signature scheme is needed. The group signing key is distributed only to writers, but the group verification key is given to all readers for the filegroup. Writers may modify files and recompute signatures using the signing key of the current time interval. Readers may check signatures on files generated at previous time intervals. We consider a model for signature schemes with lazy revocation in which the public key remains constant over time and only the signing keys change at the beginning of every time interval.

5.1. Security Definitions

Before defining signature schemes with lazy revocation, we recall the definition of signature schemes and their security under chosen-message attacks (or *CMA-security*).

Signature schemes. A signature scheme consists of three algorithms: a key generation algorithm $\mathsf{Gen}(\cdot)$ that outputs a public key/secret key pair $(\mathsf{PK},\mathsf{SK})$ (taking as input a security parameter κ), a signing algorithm $\sigma \leftarrow \mathsf{Sign}_{\mathsf{SK}}(m)$ that outputs a signature of a given message m using the signing key SK, and a verification algorithm $\mathsf{Ver}_{\mathsf{PK}}(m,\sigma)$ that outputs a bit. A signature σ is *valid* on a message m if $\mathsf{Ver}_{\mathsf{PK}}(m,\sigma) = 1$. The first two algorithms might be probabilistic, but Ver is deterministic.

The correctness property requires that $\mathsf{Ver}_{\mathsf{PK}}(m,\mathsf{Sign}_{\mathsf{SK}}(m)) = 1$, for all key pairs $(\mathsf{PK},\mathsf{SK})$ generated with the Gen algorithm and all messages m from the signature domain.

CMA-security for a signature scheme [15] requires that a polynomial-time adversary with access to a signing oracle $\mathsf{Sign}(\cdot)$ is not able to generate a message and a valid signature for which it did not query the signing oracle.

Definition of SS-LR. Signature schemes with lazy revocation include Init, Update and Derive algorithms similar to those of key-updating schemes, but with the following differences: the Init outputs also the public key of the signature scheme, and the Derive algorithm outputs directly the signing key for the time interval given as input. User keys in this case are the same as signing keys, as users perform operations only with the signing keys of the current time interval. SS-LR schemes also include signing and verification algorithms.

Definition 4 (Signature Schemes with Lazy Revocation)**.** A signature scheme with lazy revocation consists of a tuple of five polynomial-time algorithms (Init, Update, Derive, Sign, Ver) with the following properties:

- The deterministic initialization algorithm, Init, takes as input the *security parameter* 1^κ, the *number of time intervals* T, and a random seed s, and outputs an initial *trusted state* S_0 and the *public key* PK.
- The deterministic key update algorithm, Update, takes as input the current *time interval* t and the current *trusted state* S_t, and outputs a *trusted state* S_{t+1} for the next time interval.
- The deterministic key derivation algorithm, Derive, takes as input a *time interval* t and the *trusted state* S_t, and outputs a *signing key* SK_t for time interval t.
- The probabilistic signing algorithm, Sign, takes as input the *secret key* SK_t for time interval t and a *message* m, and outputs a *signature* σ.
- The deterministic verification algorithm, Ver, takes as input the *public key* PK, a *time interval* t, a *message* m and a *signature* σ and outputs a *bit*. A signature σ generated at time t is said to be *valid* on a message m if $\mathsf{Ver}(\mathsf{PK},t,m,\sigma) = 1$.

Correctness of SS-LR. Suppose that $(S_0,\mathsf{PK}) \leftarrow \mathsf{Init}(1^\kappa,T,s)$ are the public key and the initial trusted state computed from a random seed s, $S_i \leftarrow \mathsf{Update}(i,\mathsf{Update}(i-1,\ldots,\mathsf{Update}(0,S_0)\ldots))$ is the trusted state for interval $i \leq T$ and $\mathsf{SK}_i \leftarrow \mathsf{Derive}(i,S_i)$ is the signing key for interval i. The correctness property requires that $\mathsf{Ver}(\mathsf{PK},t,m,\mathsf{Sign}(\mathsf{SK}_t,m)) = 1$, for all messages m and all intervals $t \leq T$.

Security of SS-LR. The definition of security for SS-LR requires that any polynomial-time adversary with access to the signing keys SK_i for $1 \leq i \leq t$, with t adaptively chosen, and a signing oracle for time interval $t+1$ is not able to generate a message and a valid signature for time interval $t+1$ that was not obtained from the signing oracle.

Formally, consider a probabilistic polynomial-time adversary $\mathscr{A}$ that participates in the following experiment:

Initialization: Given a random seed, the initial trusted state S_0 and the public key PK are generated with the Init algorithm. PK is given to $\mathscr{A}$.

Key compromise: The adversary adaptively picks a time interval t such that $0 \leq t < T$ as follows. Starting with $t = 0,1,\ldots$, the adversary is given the signing keys M_t for all consecutive time intervals until $\mathscr{A}$ decides to output stop or t becomes equal to $T-1$.

Signature generation: $\mathscr{A}$ is given access to a signing oracle $\mathsf{Sign}(\mathsf{SK}_{t+1},\cdot)$ for time interval $t+1$ and outputs a message m and signature σ.

The adversary is successful in breaking the CMA-security of the signature scheme if m was not a query to the signing oracle and σ is a valid signature on m for time interval

$t+1$. The SS-LR scheme is CMA-secure if the adversary succeeds in this game with negligible probability.

5.2. Generic Construction from Identity-Based Signatures

We present a generic transformation of identity-based signature schemes to signature schemes with lazy revocation. We first recall identity-based signatures and their security definition, then we describe the transformation and, finally, we prove that the transformation constructs a secure signature scheme with lazy revocation.

Identity-based signatures (IBS). Identity-based signatures have been introduced by Shamir [27]. A trusted entity initially generates a *master secret key* and a *master public key*. Later the trusted entity can generate the signing key for a user from the master secret key and the user's identity, which is an arbitrary bit string. In order to verify a signature, it is enough to know the master public key and the signer's identity, which is a public string.

Definition 5 (Identity-Based Signatures)**.** An identity-based signature scheme consists of a tuple of four probabilistic polynomial-time algorithms (MKGen, UKGen, Sign, Ver) with the following properties:

- The master key generation algorithm, MKGen, takes as input the *security parameter* 1^κ, and outputs the *master public key* MPK and *master secret key* MSK of the scheme.
- The user key generation algorithm, UKGen, takes as input the *master secret key* MSK and the *user's identity* ID, and outputs the *secret key* $\mathsf{SK}_{\mathsf{ID}}$ for the user.
- The signing algorithm, Sign, takes as input the *user's secret key* $\mathsf{SK}_{\mathsf{ID}}$ and a *message m*, and outputs a *signature* σ.
- The verification algorithm, Ver, takes as input the *master public key* MPK, the signer's identity ID, a *message m* and a *signature* σ and outputs a bit. The signature σ generated by the user with identity ID is said to be *valid* on message m if $\mathsf{Ver}(\mathsf{MPK},\mathsf{ID},m,\sigma)=1$.

Correctness of IBS. The correctness property requires that, if $(\mathsf{MPK},\mathsf{MSK}) \leftarrow \mathsf{MKGen}(1^\kappa)$ is a pair of master public and secret keys for the scheme, $\mathsf{SK}_{\mathsf{ID}} \leftarrow \mathsf{UKGen}(\mathsf{MSK},\mathsf{ID})$ is the signing key for the user with identity ID, then $\mathsf{Ver}(\mathsf{MPK},\mathsf{ID},m,\mathsf{Sign}(\mathsf{SK}_{\mathsf{ID}},m))=1$, for all messages m and all identities ID.

Security of IBS. Consider a probabilistic polynomial-time adversary $\mathscr{A}$ that participates in the following experiment:

Initialization: The master public key MPK and master secret key MSK are generated with MKGen. MPK is given to $\mathscr{A}$.

Oracle queries: The adversary has access to three oracles: $\mathsf{InitID}(\cdot)$ that allows it to generate the secret key for a new identity, $\mathsf{Corrupt}(\cdot)$ that gives the adversary the secret key for an identity of its choice, and $\mathsf{Sign}(\cdot,\cdot)$ that generates the signature on a particular message and identity.

Output: The adversary outputs the identity of an uncorrupted user, a message and a signature.

The adversary succeeds in breaking the security of the IBS scheme if the signature it outputs is valid and the adversary didn't query the message to the signing oracle. The IBS scheme is secure if the adversary succeeds in this game only with negligible probability.

The transformation. We construct a signature scheme with lazy revocation from an identity-based signature scheme by letting every time interval define a different identity. Let $\mathscr{S} = (\mathsf{MKGen}, \mathsf{UKGen}, \mathsf{Sign}, \mathsf{Ver})$ be a secure identity-based signature scheme. We construct a signature scheme with lazy revocation $\mathscr{S}^{\mathtt{lr}} = (\mathsf{Init}^{\mathtt{lr}}, \mathsf{Derive}^{\mathtt{lr}}, \mathsf{Update}^{\mathtt{lr}}, \mathsf{Sign}^{\mathtt{lr}}, \mathsf{Ver}^{\mathtt{lr}})$ as follows:

- $\mathsf{Init}^{\mathtt{lr}}(1^\kappa,T)$ runs $(\mathsf{MSK},\mathsf{MPK}) \leftarrow \mathsf{MKGen}(1^\kappa)$ and outputs the initial trusted state $S_0 = \mathsf{MSK}$ and the public key MPK for the signature scheme.
- $\mathsf{Update}^{\mathtt{lr}}(t,S_t)$ outputs $S_{t+1} \leftarrow S_t$.
- $\mathsf{Derive}^{\mathtt{lr}}(t,S_t)$ runs $\mathsf{SK}_t \leftarrow \mathsf{UKGen}(S_0,t)$ and outputs SK_t.
- $\mathsf{Sign}^{\mathtt{lr}}(\mathsf{SK}_t,m)$ runs $\sigma \leftarrow \mathsf{Sign}(\mathsf{SK}_t,m)$ and outputs σ.
- $\mathsf{Ver}^{\mathtt{lr}}(\mathsf{MPK},t,m,\sigma)$ outputs the same as $\mathsf{Ver}(\mathsf{MPK},t,m,\sigma)$.

Theorem 3. *Suppose that $\mathscr{S}$ is a secure identity-based signature scheme. Then $\mathscr{S}^{\mathtt{lr}}$ is a secure signature scheme with lazy revocation.*

Proof. Correctness is easy to see. To prove security of $\mathscr{S}^{\mathtt{lr}}$, let $\mathscr{A}^{\mathtt{lr}}$ be a polynomial-time adversary successful in breaking $\mathscr{S}^{\mathtt{lr}}$. We construct an adversary $\mathscr{A}$ for $\mathscr{S}$:

- $\mathscr{A}$ is given the public key MPK of scheme $\mathscr{S}$. $\mathscr{A}$ gives MPK to $\mathscr{A}^{\mathtt{lr}}$.
- When $\mathscr{A}^{\mathtt{lr}}$ requests the secret key M_t, $\mathscr{A}$ runs $\mathsf{SK}_t \leftarrow \mathsf{Corrupt}(t)$ and gives SK_t to $\mathscr{A}^{\mathtt{lr}}$.

– When $\mathscr{A}^{\mathtt{lr}}$ makes a query m to the signing oracle for interval $t+1$, $\mathscr{A}$ runs $\sigma \leftarrow \mathsf{Sign}(t+1,m)$ and returns σ to $\mathscr{A}^{\mathtt{lr}}$.

– Finally, $\mathscr{A}^{\mathtt{lr}}$ outputs a message m and a signature σ for time interval $t+1$. Then, $\mathscr{A}$ outputs $(t+1,m,\sigma)$.

It is immediate that the probability of success of $\mathscr{A}$ is the same as the probability of success of $\mathscr{A}^{\mathtt{lr}}$ and the security of $\mathscr{S}$ implies the security of $\mathscr{S}^{\mathtt{lr}}$. □

Implementation. Generic constructions of identity-based schemes from a certain class of standard identification schemes, called *convertible*, are given by Bellare et. al. [6]. The most efficient construction of an IBS scheme is the Guillou-Quisquater scheme [16] that needs two exponentiations modulo an RSA modulus N for both generating and verifying a signature. The size of a signature is two elements of Z_N^*.

Relation to key-insulated signature schemes. A signature scheme with lazy revocation that has T time intervals can be used to construct a perfect $(T-1,T)$ key-insulated signature scheme, as defined by Dodis et al. [11]. However, the two notions are not equivalent since the attack model for key-insulated signatures is stronger. An adversary for a $(T-1,T)$ key-insulated signature scheme is allowed to compromise the signing keys for any $T-1$ intervals out of the total T intervals. Further differences between key-insulated signatures and SS-LR are that both the trusted entity and the user update their internal state at the beginning of every interval and that both parties jointly generate the signing keys for each interval.

6. Applications

In this section, we show how our cryptographic algorithms with lazy revocation can be applied to distributed cryptographic file systems, using the Plutus file system as an example. This also leads to an efficiency improvement for the revocation mechanism in Plutus.

Plutus [19] is a secure file system that uses an innovative decentralized key management scheme. In Plutus, files are divided into filegroups, each of them managed by the owner of its files. Blocks in a file are each encrypted with a different symmetric *file-block key*. The encryptions of the file-block keys for all blocks in a file are stored in a *lockbox*, which is encrypted with a *file-lockbox key*. The hash of the file is signed with a *file-signing key* for integrity protection and the signature can be verified with a *file-verification key*. The file-lockbox, file-signing and file-verification keys are the same for all files in a filegroup. Differentiation of readers and writers is done by distributing the appropriate keys to the users. In particular, the group owner distributes the file-lockbox and file-verification keys only to readers, and the file-lockbox and file-signing keys only to writers.

Plutus uses lazy revocation and a mechanism called *key rotation* for efficient key management. The file-lockbox and file-verification keys for previous time intervals can be derived from the most recent keys. Our cryptographic primitives with lazy revocation generalize the key rotation mechanism because we allow previous keys to be derived from our user key, which may be different from the actual key used for cryptographic operations at the current time interval. This allows more flexibility in constructing key-updating schemes.

We now recall the Plutus key rotation mechanisms for encryption and signing keys and demonstrate in both cases how our cryptographic primitives with lazy revocation lead to more efficient solutions.

For *encryption*, the group manager as the trusted entity uses the inverse of the RSA trapdoor permutation to update the file-lockbox encryption key after every user revocation. Users derive file-lockbox keys of previous time intervals using the public RSA trapdoor permutation. The construction does not have a cryptographic security proof and cannot be generalized to arbitrary trapdoor permutations because the output of the trapdoor permutation is not necessarily uniformly distributed. But it could be fixed by applying a hash function to the output of the trapdoor permutation for deriving the key, which makes the construction provably secure in the random oracle model [2]. Indeed, this is our *trapdoor permutation* key-updating scheme [2].

However, the *binary-tree key-updating scheme* [2] is more efficient because it uses only symmetric-key operations (e.g., a block cipher). Used in a symmetric encryption scheme with lazy revocation according to Section 3, it improves the time for updating and deriving file-lockbox keys by several orders of magnitude.

For *signatures*, Plutus uses RSA in a slightly different method than for encryption. A different public-key/secret-key pair is generated by the group owner after every revocation, and hence the RSA moduli differ for all time intervals and need to be stored with the file meta-data. The public verification exponent can be derived from the file-lockbox key by readers. An alternative solution based on our signature schemes with lazy revocation according to Section 5 uses only one verification key and achieves two distinct advantages: first, the storage space for the public keys is reduced to a constant from linear in the number of revocations and, secondly, the expensive operation of deriving the public verification exponent in Plutus does not need to be performed. For example, using the Guillou-Quisquater IBS scheme, deriving the public key of a time interval during verification takes only a few hash function applications.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," in *Proc. 5th Symposium on Operating System Design and Implementation (OSDI)*, Usenix, 2002.

[2] M. Backes, C. Cachin, and A. Oprea, "Secure key-updating for lazy revocation," Technical Report RZ 3627, IBM Research, Aug. 2005.

[3] M. Bellare, R. Canetti, and H. Krawczyk, "Keyed hash functions for message authentication," in *Proc. Crypto 1996*, vol. 1109 of *Lecture Notes in Computer Science*, pp. 1–15, Springer-Verlag, 1996.

[4] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *Proc. 38th Symposium on Foundations of Computer Science (FOCS)*, pp. 394–403, IEEE, 1997.

[5] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," in *Proc. Crypto 1994*, vol. 839 of *Lecture Notes in Computer Science*, pp. 341–358, Springer-Verlag, 1994.

[6] M. Bellare, C. Namprempre, and G. Neven, "Security proofs for identity-based identification and signature schemes," in *Proc. Eurocrypt 2004*, vol. 3027 of *Lecture Notes in Computer Science*, pp. 268–286, Springer-Verlag, 2004.

[7] M. Blaze, "A cryptographic file system for Unix," in *Proc. First ACM Conference on Computer and Communication Security (CCS)*, pp. 9–16, 1993.

[8] M. Blaze, "Key management in an encrypting file system," in *Proc. Summer 1994 USENIX Technical Conference*, pp. 28–35, 1994.

[9] R. Canetti, S. Halevi, and J. Katz, "A forward-secure public-key encryption scheme," in *Proc. Eurocrypt 2003*, vol. 2656 of *Lecture Notes in Computer Science*, pp. 255–271, Springer-Verlag, 2003.

[10] G. Cattaneo, L. Catuogno, A. D. Sorbo, and P. Persiano, "The design and implementation of a transparent cryptographic file system for Unix," in *Proc. USENIX Annual Technical Conference 2001, Freenix Track*, pp. 199–212, 2001.

[11] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Strong key-insulated signature schemes," in *Proc. 6th International Workshop on Theory and Practice in Public Key Cryptography (PKC)*, vol. 2567 of *Lecture Notes in Computer Science*, pp. 130–144, Springer-Verlag, 2003.

[12] K. Fu, "Group sharing and random access in cryptographic storage file systems," Master's thesis, Massachusetts Institute of Technology, 1999.

[13] K. Fu, S. Kamara, and T. Kohno, "Key regression: Enabling efficient key distribution for secure distributed storage." Cryptology ePrint Archive, Report 2005/303, 2005.

[14] E. Goh, H. Shacham, N. Modadugu, and D. Boneh, "SiRiUS: Securing remote untrusted storage," in *Proc. Network and Distributed Systems Security (NDSS) Symposium 2003*, pp. 131–145, ISOC, 2003.

[15] S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing*, vol. 17, no. 2, pp. 281–308, 1988.

[16] L. Guillou and J. Quisquater, "A "paradoxical" identity-based signature scheme resulting from zero-knowledge," in *Proc. Crypto 1988*, vol. 403 of *Lecture Notes in Computer Science*, pp. 216–231, Springer-Verlag, 1988.

[17] S. Halevi and P. Rogaway, "A tweakable enciphering mode," in *Proc. Crypto 2003*, vol. 2729 of *Lecture Notes in Computer Science*, pp. 482–499, Springer-Verlag, 2003.

[18] M. Jakobsson, "Fractal hash sequence representation and traversal," in *Proc. IEEE International Symposium on Information Theory*, IEEE, 2002.

[19] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proc. Second USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[20] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proc. 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 190–201, ACM, 2000.

[21] J. Li, M. Krohn, D. Mazieres, and D. Shasha, "Secure untrusted data repository," in *Proc. 6th Symposium on Operating System Design and Implementation (OSDI)*, pp. 121–136, Usenix, 2004.

[22] M. Liskov, R. Rivest, and D. Wagner, "Tweakable block ciphers," in *Proc. Crypto 2002*, vol. 2442 of *Lecture Notes in Computer Science*, pp. 31–46, Springer-Verlag, 2002.

[23] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.

[24] E. Miller, D. Long, W. Freeman, and B. Reed, "Strong security for distributed file systems," in *Proc. First USENIX Conference on File and Storage Technologies (FAST)*, pp. 1–13, 2002.

[25] D. Naor, A. Shenhav, and A. Wool, "Toward securing untrusted storage without public-key operations," in *Proc. First International Workshop on Storage Security and Survivability (StorageSS)*, ACM, 2005.

[26] E. Riedel, M. Kallahalla, and R. Swaminathan, "A framework for evaluating storage system security," in *Proc. First USENIX Conference on File and Storage Technologies (FAST)*, pp. 15–30, 2002.

[27] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proc. Crypto 1984*, vol. 196 of *Lecture Notes in Computer Science*, pp. 47–53, Springer-Verlag, 1985.