Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage

Kevin Fu
U. Mass. Amherst

Seny Kamara
Johns Hopkins University

Tadayoshi Kohno
U.C. San Diego

Abstract

The Plutus file system introduced the notion of key rotation as a means to derive a sequence of temporally-related keys from the most recent key. In this paper we show that, despite natural intuition to the contrary, key rotation schemes cannot generically be used to key other cryptographic objects; in fact, keying an encryption scheme with the output of a key rotation scheme can yield a composite system that is insecure. To address these shortcomings, we introduce a new cryptographic object called a key regression scheme, and we propose three constructions that are provably secure under standard cryptographic assumptions. We implement key regression in a secure file system and empirically show that key regression can significantly reduce the bandwidth requirements of a content publisher under realistic workloads using lazy revocation. Our experiments also serve as the first empirical evaluation of either a key rotation or key regression scheme.

Keywords: *Key regression, key rotation, lazy revocation, key distribution, content distribution network, hash chain, security proofs.*

1 Introduction

Content distribution networks (CDNs) such as Akamai [3], BitTorrent [14], and Coral [20] enable *content publishers* with low-bandwidth connections to make single-writer, many-reader content available at high throughput. When a CDN is untrusted and the content publisher cannot rely on the network to enforce proper access control, the content publisher can achieve access control by encrypting the content and distributing the cryptographic keys to legitimate users [22, 25, 30, 32, 39, 42]. Under the *lazy revocation* model for access control [22, 32], following the *eviction* of a user from the set of members, the content publisher will encrypt future content with a new cryptographic key and will, upon request, distribute that new key to all remaining and future members. The content publisher does not immediately re-encrypt all pre-existing content since the evicted member could have already cached that content.

The content publisher can use the CDN to distribute the encrypted content, but without the aid of a trusted server, the content publisher must distribute all the cryptographic keys to members directly. To prevent the publisher's connection from becoming a bottleneck, the Plutus file system [32] introduced a new cryptographic object called a *key rotation scheme*. Plutus uses the symmetric key K_i to encrypt content during the i-th time period, e.g., before the i-th eviction. If a user becomes a member during the i-th time period, then Plutus gives that member the i-th key K_i. From [32], the critical properties of a key rotation scheme are that given the i-th key K_i it is (1) easy to compute the keys K_j for all previous time periods $j < i$, but (2) computationally infeasible to compute the keys K_l for future time periods $l > i$. Property (1) enables the content publisher to transfer only a single small key K_i to new members wishing to access all current and past content, rather than the potentially large set of keys $\{K_1, K_2, \ldots, K_i\}$; this property reduces the bandwidth requirements on the content publisher. Property (2) is intended to prevent a member evicted during the i-th time period from accessing (learning the contents of) content encrypted during the l-th time period, $l > i$.

1.1 Overview of contributions

In this work we uncover a design flaw with the definition of a key rotation scheme. To address the deficiencies with key rotation, we introduce a new cryptographic object called a *key regression scheme*. We present RSA-based, SHA1-based, and AES-based key regression schemes. We implement and analyze the performance of key regression in the context of a secure file system. The following paragraphs summarize our contributions in more detail.

Negative results on key rotation. We begin by presenting a design flaw with the definition of key rotation: for any realistic key rotation scheme, even though a member evicted during the i-th time period *cannot predict*

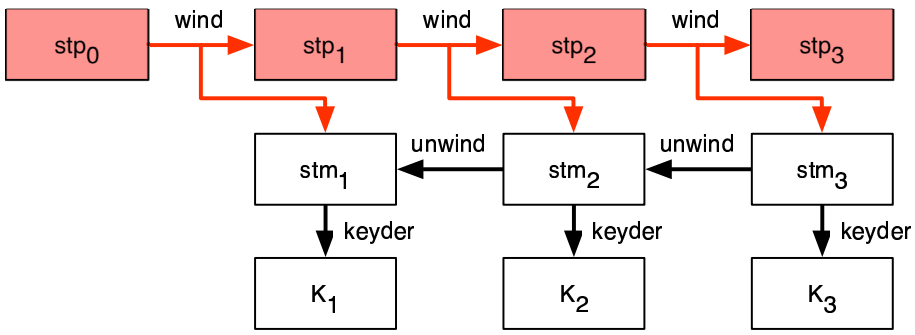

Figure 1. Key regression overview; stp_i and stm_i respectively represent the i-th publisher and member states.

(except with negligible probability) subsequent keys K_l, $l > i$, the evicted member *can distinguish* subsequent keys K_l from random. The lack of pseudorandomness follows from the fact that if an evicted member is given the real key K_l, then by definition (i.e., by property (1)) the evicted member can recover the real key K_i; but given a random key instead of K_l, the evicted member will with high probability recover a key $K_i' \neq K_i$. The difference between unpredictability and lack of pseudorandomness can have severe consequences in practice. To illustrate the seriousness of this design flaw, we describe a key rotation scheme and a symmetric encryption scheme that individually meet their desired security properties (property (2) for key rotation and IND-CPA privacy for symmetric encryption [7]), but when combined (e.g., when a content publisher uses the keys from the key rotation scheme to key the symmetric encryption scheme) result in a system that fails to provide even a weak form of privacy.[1]

Fixing key rotation with key regression. While the above counter example does not imply that all systems employing key rotation will fail just as drastically, it does motivate finding a key rotation-like object that still achieves property (1) (or something similar) but (property (2′)) produces future keys that are pseudorandom to evicted members (as opposed to just unpredictable). Assuming the new object achieves pseudorandomness, one could use it as a black box to key other cryptographic constructs without worrying about the resulting system failing as drastically as the one described above. A *key regression scheme* is such a key rotation-like object.

To describe key regression, we must enact a paradigm shift: rather than give a new member the i-th key K_i directly, the content publisher would give the member a *member state* stm_i. From the member state, the member could derive the encryption key K_i for the i-th time period, as well as all previous member states stm_j, $j < i$. By transitivity, a member given the i-th member state could also derive all previous keys K_j. By separating the member states from the keys, we can build key regression schemes where the keys K_l, $l > i$, are pseudorandom to evicted members possessing only the i-th member state stm_i. Intuitively, the trick that we use in our constructions to make the keys K_l pseudorandom is to ensure that given both K_l and stm_i, it is still computationally infeasible for the evicted member to compute the l-th member state stm_l. Viewed another way, there is no path from K_l to stm_i in Figure 1 and vice-versa.

Our constructions. We refer to our three preferred key regression schemes as KR-RSA, KR-SHA1, and KR-AES. Rather than rely solely on potentially error-prone heuristic methods for analyzing the security of our constructions, we prove under reasonable assumptions that all three are secure key regression schemes. Our security proofs use the reduction-based provable security approach pioneered by Goldwasser and Micali [27] and lifted to the concrete setting by Bellare, Kilian, and Rogaway [8]. For KR-RSA, our proof is based on the assumption that RSA is one-way. For the proof of both KR-RSA and KR-SHA1, we assume that SHA1 is a random oracle [9]. For the proof of KR-AES, we assume that AES is a secure pseudorandom permutation [8, 35].

Implementation and evaluation. We integrated key regression into a secure file system to measure the performance characteristics of key regression in a real application. Our measurements show that key regression can significantly reduce the bandwidth requirements of a publisher distributing decryption keys to members. On a simulated cable modem, a publisher using key regression can distribute 1 000 keys to 181 clients/sec

[1] We stress that the novelty here is in identifying the design flaw with key rotation, not in presenting a specific counter example. Indeed, the counter example follows naturally from our observation that a key rotation scheme does not produce pseudorandom keys.

whereas without key regression the cable modem limits the publisher to 20 clients/sec. The significant gain in throughput conservation comes at no observable cost to client latency, even though key regression requires more client-side computation. Our measurements show that key regression actually reduces client latency in cases of highly dynamic group membership. Our study represents the first empirical measurements of either a key regression or key rotation scheme.

Contrary to conventional wisdom, on our testbed we find that KR-AES can perform more than four times as many unwinds/sec than KR-SHA1. Our measurements can assist developers in selecting the most appropriate key regression scheme for particular applications.

Applications. Key regression benefits publishers of popular content who have limited bandwidth to their trusted servers, or who may not always be online, but who can use an untrusted CDN to distribute encrypted content at high throughput. Our experimental results show that a publisher using key regression on a low-bandwidth connection can serve more clients than the strawman approach of having the publisher distribute all keys $\{K_1, K_2, \ldots, K_i\}$ directly to members. Moreover, our experimental results suggest that key regression can be significantly better than the strawman approach when i is large, as might be the case if the publisher has a high membership turnover rate. Such a publisher might be an individual, a startup, or a cooperative with popular content but with few network resources. The possibilities for such content range from blogs and amateur press to operating systems and various forms of multimedia. To elaborate on one such form of content, operating systems, Mandriva Linux currently uses the BitTorrent CDN to distribute its latest Linux distributions to its Mandriva Club members [37]. Mandriva controls access to these distributions by only releasing the `.torrent` files to its members. Using key regression and encryption for access control, Mandriva could exercise finer-grained access control over its distributions, allowing members through time period i to access all versions of the operating system including patches, minor revisions and new applications added through time period i, but no additions to the operating system after the i-th time period.[2]

Versions. This is an extended abstract. The full version of this paper appears on the IACR ePrint Archive [24]. Part of this work also appears as Chapter 4 of [22].

[2]While Mandriva may wish to exercise access control over non-security-critical patches and upgrades, Mandriva would likely wish to allow all Mandriva users, including evicted Mandriva Club members, access to all security-critical patches. To enable such access, Mandriva could encrypt all security-critical patches with the key for the time period to which the patch is first applicable, or Mandriva could simply not encrypt security-critical patches.

1.2 Related work

The key rotation scheme in Plutus [32] inspired our research in key regression. Bellare and Yee [10] introduce the notion of a forward-secure pseudorandom bit generator (FSPRG). One can roughly view forward-secure pseudorandom bit generation as the mirror image of key regression. Whereas a key regression scheme is designed to prevent an evicted member in possession of stm_i from distinguishing *subsequent* encryption keys K_l, $l > i$, from random, a FSPRG is designed to prevent an adversary who learns the state of the FSPRG at some point in time from distinguishing *previous* outputs of the FSPRG from random. In our security proof for KR-AES, we make the relationship between key regression and FSPRGs concrete by first proving that one can build a secure key regression scheme from any secure FSPRG by essentially running the FSPRG backwards. Abdalla and Bellare formally analyze methods for rekeying symmetric encryption schemes [1], and one of their constructions is a FSPRG.

As pointed out by Boneh et al. [13], one possible mechanism for distributing updated content encryption keys for a secure file system is to use a broadcast encryption scheme [17, 18, 19, 40]. Indeed, one of the main challenges faced by an encrypted file system is the distribution of the encryption keys to the remaining (not evicted) set of users, and broadcast encryption provides an ideal solution. We note, however, that key distribution is orthogonal to the specific problem addressed by key regression; a key regression scheme is a key *generation* algorithm as opposed to a key *distribution* algorithm. Key regression simply assumes the existence of a secure distribution channel, of which broadcast encryption is one possible instantiation. Self-healing key distribution with revocation [48] protocols are resilient even when broadcasts are lost on the network. One can view key regression as having the self-healing property in perpetuity.

In concurrent work, and also motivated by the key rotation scheme in Plutus [32], Backes, Cachin, and Oprea formalize the notion of *key-updating for lazy revocation schemes* [6] and consider the composition of key-updating for lazy revocation schemes with other cryptographic objects [5]. The notion of a key-updating for lazy revocation scheme in [6] is essentially identical to our notion of a key regression scheme. Using our parlance, in [6] they also propose several ways of building key regression schemes; one of their proposals is identical to our KR-PRG construction (Construction 7.3), and another proposal is a natural extension of our construction KR-RSA-RO (Construction 10.1). Although we remark on the existence of a tree-based key regression scheme in Section 5, [6] take the idea of a tree-based key regression scheme further by formally defining and

proving the security of a slightly different tree-based construction. In [6] the authors also observe that one can use the keys output by a key regression scheme as the randomness source for the setup algorithm of a (possibly different) key regression scheme; this observation enables the composition of multiple key regression schemes.

2 Notation

If x and y are strings, then $|x|$ denotes the length of x in bits and $x\|y$ denotes their concatenation. If x and y are two variables, we use $x \leftarrow y$ to denote the assignment of the value of y to x. If Y is a set, we denote the selection of a random element in Y and its assignment to x as $x \xleftarrow{\$} Y$. If f is a deterministic (resp., randomized) function, then $x \leftarrow f(y)$ (resp., $x \xleftarrow{\$} f(y)$) denotes the process of running f on input y and assigning the result to x. We use the special symbol $\perp$ to denote an error.

We use $\text{AES}_K(M)$ to denote the process of running the AES block cipher with key K on input block M. We use $\text{SHA1}(M)$ to denote the process of running the SHA1 hash function on input M. An RSA [43] key generator for some security parameter k is a randomized algorithm $\mathcal{K}_{\text{rsa}}$ that returns a triple (N, e, d). Since our analyses are in the concrete setting, we write $(N, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$ rather than $(N, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}(k)$. The modulus N is the product of two distinct odd primes p, q such that $2^{k-1} \leq N < 2^k$; the encryption exponent $e \in \mathbb{Z}^*_{\varphi(N)}$ and the decryption exponent $d \in \mathbb{Z}^*_{\varphi(N)}$ are such that $ed \equiv 1 \bmod \varphi(N)$, where $\varphi(N) = (p-1)(q-1)$. Section 10 describes what it means for an RSA key generator to be one-way.

3 Problems with key rotation

A key rotation scheme [32] consists of three algorithms: setup, wndkey, and unwndkey. Figure 2 shows the original (RSA-based) Plutus key rotation scheme [32]. Following Plutus, and as Naor, Shenhav, and Wool also observe [41], one familiar with hash chains [34] and S/KEY [29] might design the key rotation scheme in Figure 3. Such a scheme is more efficient than the scheme in Figure 2, but is limited because it can only produce MW ("max wind") keys, where MW is a parameter chosen by the implementor or at configuration time. A content publisher runs the setup algorithm to initialize a key rotation scheme; the result is public information pk for all users and a secret sk_1 for the content publisher. The content publisher invokes $\text{wndkey}(\mathsf{sk}_i)$ to obtain the key K_i and a new secret sk_{i+1}. Any user in possession of K_i, $i > 1$, and pk can invoke $\text{unwndkey}(K_i, \mathsf{pk})$ to obtain K_{i-1}. Informally, the desired security property of a key rotation scheme is that, given only K_i and pk, it should be computationally infeasible for an evicted member (the adversary) to compute K_l, for any $l > i$. The Plutus construction in Figure 2 has this property under the RSA one-wayness assumption (defined in Section 10), and the construction in Figure 3 has this property if one replaces SHA1 with a random oracle [9].

The problem. In Section 1 we observed that the l-th key output by a key rotation scheme cannot be pseudorandom, i.e., will be distinguishable from a random string, to an ex-member in possession of the key K_i for some previous time period $i < l$.[3] We consider the following example to emphasize how this lack of pseudorandomness might impact the security of a real system that combines a key rotation scheme and a symmetric encryption scheme as a black box.

For our example, we first present a key rotation scheme $\overline{\mathcal{KO}}$ and an encryption scheme $\overline{\mathcal{SE}}$ that individually satisfy their respective security goals (unpredictability for the key rotation scheme and IND-CPA privacy [7] for the symmetric encryption scheme). To build $\overline{\mathcal{KO}}$, we start with a secure key rotation scheme $\mathcal{KO}$; $\overline{\mathcal{KO}}$ outputs keys twice as long as $\mathcal{KO}$. The $\overline{\mathcal{KO}}$ winding algorithm $\overline{\text{wndkey}}$ invokes $\mathcal{KO}$'s winding algorithm to obtain a key K; $\overline{\text{wndkey}}$ then returns $K\|K$ as its key. On input a key $K\|K$, $\overline{\text{unwndkey}}$ invokes $\mathcal{KO}$'s unwinding algorithm with input K to obtain a key K'; $\overline{\text{unwndkey}}$ then returns $K'\|K'$ as its key. If the keys output by wndkey are unpredictable to evicted members, then so must be the keys output by $\overline{\text{wndkey}}$. To build $\overline{\mathcal{SE}}$, we start with a secure symmetric encryption scheme $\mathcal{SE}$; $\overline{\mathcal{SE}}$ uses keys that are twice as long as $\mathcal{SE}$. The $\overline{\mathcal{SE}}$ encryption and decryption algorithms take the key K, split it into two halves $K = L_1\|L_2$, and run the respective algorithms of $\mathcal{SE}$ with key $L_1 \oplus L_2$. If the key K is random, then the key $L_1 \oplus L_2$ is random and $\overline{\mathcal{SE}}$ runs the $\mathcal{SE}$ encryption algorithm with a uniformly selected random key. This means that $\overline{\mathcal{SE}}$ satisfies the standard IND-CPA security goal if $\mathcal{SE}$ does.

Despite the individual security of both $\overline{\mathcal{KO}}$ and $\overline{\mathcal{SE}}$, when the keys output by $\overline{\mathcal{KO}}$ are used to key $\overline{\mathcal{SE}}$, $\overline{\mathcal{SE}}$ will always run $\mathcal{SE}$ with the all-zero key; i.e., the content publisher will encrypt all content under the same constant key. An adversary can thus trivially compromise the privacy of all encrypted data, including data

[3] Technically, there may be pathological examples where the l-th key is pseudorandom to a member given the i-th key, but these examples seem to have other problems of their own. For example, consider a key rotation scheme like the one in Figure 3, but where SHA1 is replaced with a function mapping all inputs to some constant string C, e.g., the all 0 key. Now set $\mathsf{MW} = 2$, $i = 1$, and $l = 2$. In this pathological example K_2 is clearly random to the evicted member, meaning (better than) pseudorandom. But this construction still clearly lacks our desired pseudorandomness property since the key K_1 is always the constant string C.

Alg. setup
$(N, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$; $K \xleftarrow{\$} \mathbb{Z}_N^*$
$\mathsf{pk} \leftarrow \langle N, e\rangle$; $\mathsf{sk} \leftarrow \langle K, N, d\rangle$
Return $(\mathsf{pk}, \mathsf{sk})$

Alg. wndkey($\mathsf{sk} = \langle K, N, d\rangle$)
$K' \leftarrow K^d \bmod N$
$\mathsf{sk}' \leftarrow \langle K', N, d\rangle$
Return (K, sk')

Alg. unwndkey($K, \mathsf{pk} = \langle N, e\rangle$)
Return $K^e \bmod N$

Figure 2. The Plutus key rotation scheme; $\mathcal{K}_{\text{rsa}}$ is an RSA key generator.

Alg. setup
$K_{\mathsf{MW}} \xleftarrow{\$} \{0,1\}^{160}$; $\mathsf{pk} \leftarrow \varepsilon$
For $i = \mathsf{MW}$ downto 2 do
$K_{i-1} \leftarrow \text{SHA1}(K_i)$
$\mathsf{sk} \leftarrow \langle 1, K_1, \ldots, K_{\mathsf{MW}}\rangle$
Return $(\mathsf{pk}, \mathsf{sk})$

Alg. wndkey($\mathsf{sk} = \langle i, K_1, \ldots, K_{\mathsf{MW}}\rangle$)
If $i > \mathsf{MW}$ return $(\perp, \mathsf{sk})$
$\mathsf{sk}' \leftarrow \langle i+1, K_1, \ldots, K_{\mathsf{MW}}\rangle$
Return (K_i, sk')

Alg. unwndkey(K, pk)
// ignore pk
$K' \leftarrow \text{SHA1}(K)$
Return K'

Figure 3. A hash chain-based key rotation scheme.

encrypted during time periods $l > i$ after being evicted. Although the construction of $\overline{\mathcal{KO}}$ and $\overline{\mathcal{SE}}$ may seem somewhat contrived (though we hope less contrived than some other possible counter examples), this example shows that combining a key rotation scheme and an encryption scheme may have undesirable consequences and, therefore, that it is not wise to use (even a secure) key rotation scheme as a black box to directly key other cryptographic objects.

4 Key Regression

The negative result in Section 3 motivates our quest to find a new cryptographic object, similar to key rotation, but for which the keys generated at time periods $l > i$ are pseudorandom to any adversary evicted at time i. Here we formalize such an object: a key regression scheme. Following the reduction-based practice-oriented provable security approach [8, 27], our formalisms involve carefully defining the syntax, correctness requirements, and security goal of a key regression scheme. These formalisms enable us to, in Sections 8–10, prove that our preferred constructions are secure under reasonable assumptions. We desire provable security over solely *ad hoc* analyses since, under *ad hoc* methods alone, one can never be completely convinced that a cryptographic construction is secure even if one assumes that the underlying components (e.g., block ciphers, hash functions, RSA) are secure.

Overview of key regression. Figure 1 gives an abstract overview of a key regression scheme. The content publisher has content publisher states stp_i from which it derives future publisher and member states. When using a key regression scheme, instead of giving a new member the i-th key K_i, the content publisher would give the member the i-th member state stm_i. As the arrows in Figure 1 suggest, given stm_i, a member can efficiently compute all previous member states and the keys $K_1, \ldots, K_i$. Although it would be possible for an ex-member to distinguish future member states stm_l, $l > i$, from random (the ex-member would extend our observation on the lack of pseudorandomness in key rotation schemes), because there is no efficient path between the future keys K_l and the ex-member's last member state stm_i, it is possible for a key regression scheme to produce future keys K_l that are pseudorandom (indistinguishable from random). We present some such constructions in Section 5.

On an alternative: Use key rotation carefully. Figure 1 might suggest an alternative approach for fixing the problems with key rotation. Instead of using the keys K_i from a key rotation scheme to directly key other cryptographic objects, use a function of K_i, like $\text{SHA1}(K_i)$, instead. If one models SHA1 as a random oracle and if the key rotation scheme produces unpredictable future keys K_l, then it might seem reasonable to conclude that an ex-member given K_i should not be able to distinguish future values $\text{SHA1}(K_l)$, $l > i$, from random. While this reasoning may be sound for some specific key rotation schemes (this reasoning actually serves as the basis for our derivative of the construction in Figure 2, KR-RSA in Construction 5.3) we dislike this approach for several reasons. First, we believe that it is unreasonable to assume that every engineer will know to or remember to use the hash function. Further, even if the engineer knew to hash the keys, the engineer might not realize that simply computing $\text{SHA1}(K_l)$ may not work with all key rotation schemes, which means that the engineer cannot use a key rotation scheme as a black box. For example, while $\text{SHA1}(K_l)$ would work for the scheme in Figure 2, it would cause problems for the scheme in Figure 3. We choose to consider a new

cryptographic object, key regression, because we desire a cryptographic object that is not as prone to accidental misuse. Additionally, by focusing attention on a new cryptographic object, we allow ourselves greater flexibility in how we construct objects that meet our requirements. For example, one of our preferred constructions (KR-AES, Construction 5.2) does not use a hash function and is therefore secure in the standard model instead of the random oracle model; see also KR-FSPRG (Construction 6.1) and KR-PRG (Construction 7.3).

4.1 Syntax and correctness requirements

Syntax. Here we formally define the syntax of a key regression scheme $\mathcal{KR} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$. Let H be a random oracle; for notational consistency, all four algorithms are given access to the random oracle, though the algorithms for some constructions may not use the random oracle in their computations. Via $\mathsf{stp} \xleftarrow{\$} \mathsf{setup}^H$, the randomized setup algorithm returns a publisher state. Via $(\mathsf{stp}', \mathsf{stm}) \xleftarrow{\$} \mathsf{wind}^H(\mathsf{stp})$, the randomized winding algorithm takes a publisher state stp and returns a pair of publisher and member states or the error code $(\perp, \perp)$. Via $\mathsf{stm}' \leftarrow \mathsf{unwind}^H(\mathsf{stm})$ the deterministic unwinding algorithm takes a member state stm and returns a member state or the error code $\perp$. Via $K \leftarrow \mathsf{keyder}^H(\mathsf{stm})$ the deterministic key derivation algorithm takes a member state stm and returns a key $K \in \mathsf{DK}$, where DK is the *derived key space* for $\mathcal{KR}$. Let $\mathsf{MW} \in \{1, 2, \ldots\} \cup \{\infty\}$ denote the maximum number of derived keys that $\mathcal{KR}$ is designed to produce. We do not define the behavior of the algorithms when input the error code $\perp$.

Correctness. Our first correctness criterion for a key regression scheme is that the first MW times that wind is invoked, it always outputs valid member states, i.e., the outputs are never $\perp$. Our second correctness requirement ensures that if stm_i is the i-th member state output by wind, and if $i > 1$, then from stm_i, one can derive all previous member states stm_j, $0 < j < i$. Formally, let $\mathsf{stp}_0 \xleftarrow{\$} \mathsf{setup}$ and, for $i = 1, 2, \ldots$, let $(\mathsf{stp}_i, \mathsf{stm}_i) \xleftarrow{\$} \mathsf{wind}^H(\mathsf{stp}_{i-1})$. For each $i \in \{1, 2, \ldots, \mathsf{MW}\}$, we require that $\mathsf{stm}_i \neq \perp$ and that, for $i \geq 2$, $\mathsf{unwind}^H(\mathsf{stm}_i) = \mathsf{stm}_{i-1}$.

Remarks. Although we allow wind to be randomized, the wind algorithms in all of our constructions are deterministic. We allow wind to return $(\perp, \perp)$ since we only require that wind return non-error states for its first MW invocations. We use the pair $(\perp, \perp)$, rather than simply $\perp$, to denote an error from wind since doing so makes our pseudocode cleaner. We allow unwind to return $\perp$ since the behavior of unwind may be undefined when input the first member state. A construction may use multiple random oracles, but since one can always obtain multiple random oracles from a single random oracle [9], our definitions assume just one. It is straightforward to modify our syntax, correctness requirements, and (subsequent) security definition to accommodate key regression schemes for which the random oracle depends on the output of setup. We stress that MW is a correctness parameter of $\mathcal{KR}$, not a security parameter, meaning that even though the correctness criteria must hold for MW invocations of wind, the security goal may not. One can also further generalize our definition and allow unwind and keyder to be randomized, though we do not envision such constructions in practice.

4.2 Security goal

For security, we desire that if a member (adversary) is evicted during the i-th time period, then the adversary will not be able to distinguish the keys derived from any subsequent member state stm_l, $l > i$, from randomly selected keys. Definition 4.1 captures this goal as follows. We allow the adversary to obtain as many member states as it wishes (via a WindO oracle). The WindO oracle returns only a member state rather than both a member and publisher state. Once the adversary is evicted, its goal is to break the pseudorandomness of subsequently derived keys. To model this, we allow the adversary to query a key derivation oracle KeyderO. The key derivation oracle will either return real derived keys (via internal calls to wind and keyder) or random keys. The adversary's goal is to guess whether the KeyderO oracle's responses are real derived keys or random keys.

Definition 4.1 [Security for key regression schemes.] Let $\mathcal{KR} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ be a key regression scheme. Let $\mathcal{A}$ be an adversary. Consider the experiments $\mathbf{Exp}^{\text{kr-}b}_{\mathcal{KR},\mathcal{A}}$, $b \in \{0, 1\}$, and the oracles WindO and $\mathsf{KeyderO}_b$ below. The adversary runs in two stages, member and non-member, and returns a bit.

Experiment $\mathbf{Exp}^{\text{kr-}b}_{\mathcal{KR},\mathcal{A}}$
  Pick random oracle H
  $i \leftarrow 0$; $\mathsf{stp} \xleftarrow{\$} \mathsf{setup}^H$
  $\mathsf{st} \xleftarrow{\$} \mathcal{A}^{\mathsf{WindO},H}(\mathsf{member})$
  $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{KeyderO}_b,H}(\mathsf{non\text{-}member}, \mathsf{st})$
  Return b'
Oracle WindO
  $i \leftarrow i + 1$; If $i > \mathsf{MW}$ then return $\perp$
  $(\mathsf{stp}, \mathsf{stm}) \xleftarrow{\$} \mathsf{wind}^H(\mathsf{stp})$
  Return stm
Oracle $\mathsf{KeyderO}_b$
  $i \leftarrow i + 1$; If $i > \mathsf{MW}$ then return $\perp$
  $(\mathsf{stp}, \mathsf{stm}) \xleftarrow{\$} \mathsf{wind}^H(\mathsf{stp})$

If $b = 1$ then $K \leftarrow \mathsf{keyder}^H(\mathsf{stm})$
If $b = 0$ then $K \xleftarrow{\$} \mathsf{DK}$
Return K

The *KR-advantage* of $\mathcal{A}$ in breaking the security of $\mathcal{KR}$ is defined as

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} = \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1\right] - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1\right].$$

Under the concrete security approach [8], we say that $\mathcal{KR}$ is *KR-secure* if for any adversary $\mathcal{A}$ attacking $\mathcal{KR}$ with resources (running time, size of code, number of oracle queries) limited to "practical" amounts, the KR-advantage of $\mathcal{A}$ is "small." ▮

Remarks. Since the publisher is in charge of winding and is not supposed to invoke the winding algorithm more than the prescribed maximum number of times, MW, the WindO and KeyderO oracles in our security definition only respond to the first MW queries from the adversary. Alternatively, we could remove the conditional check for $i >$ MW in the pseudocode for WindO and KeyderO and instead ask that the underlying wind algorithm behave appropriately if invoked more than MW times, e.g., by maintaining the counter internally. Since a key regression scheme will have multiple recipients of member keys, we must consider coalitions of adversaries; i.e., can two or more adversaries collude to obtain additional information? Because of the property that given any member state one can derive all previous member states, multiple colluding adversaries cannot obtain more information than a single adversary who makes the most WindO and KeyderO oracle queries. In addition to desiring that future derived keys be pseudorandom to evicted members, we desire that all the derived keys be pseudorandom to adversaries that are never members. If a key regression scheme is secure under Definition 4.1, then the key regression scheme also satisfies this weaker security goal since one can view adversaries that are never members as adversaries that make zero WindO oracle queries. Unlike with key rotation schemes (Section 3), the pseudorandomness of future keys means that a content publisher can use the keys output by a secure key regression scheme to key other cryptographic objects like symmetric encryption schemes [7] and MACs [8]; as [1, 10] do for rekeying schemes and FSPRGs, [5] makes this reasoning formal for key regression schemes.

5 Our preferred constructions

We are now in a position to describe our three preferred key regression schemes, KR-SHA1, KR-AES and KR-RSA. Table 1 summarizes some of their main properties. KR-SHA1 is a derivative of the key rotation scheme in Figure 3 and KR-RSA is a derivative of the Plutus key rotation scheme in Figure 2. The primary differences between the new key regression schemes KR-SHA1 and KR-RSA and the original key rotation schemes are the addition of the new, SHA1-based keyder algorithms and the adjusting of terminology (e.g., member states in these key regression schemes correspond to keys in the original key rotation schemes). KR-AES is new but is based on one of Bellare and Yee's forward-secure pseudorandom bit generators (FSPRGs) [10].

5.1 The KR-SHA1 construction

Construction 5.1 details our KR-SHA1 construction. In the construction of KR-SHA1, we prepend the string 0^8 to the input to SHA1 in keyder to ensure that the inputs to SHA1 never collide between the keyder and unwind algorithms; note that the stm variable always denotes a 160-bit string.

Construction 5.1 [KR-SHA1.] The key regression scheme KR-SHA1 $=$ (setup, wind, unwind, keyder) is defined as follows. MW is a positive integer and a parameter of the construction.

Alg. setup
  $\mathsf{stm}_{\mathsf{MW}} \xleftarrow{\$} \{0,1\}^{160}$
  For $i = \mathsf{MW}$ downto 2 do
    $\mathsf{stm}_{i-1} \leftarrow \mathsf{unwind}(\mathsf{stm}_i)$
  $\mathsf{stp} \leftarrow \langle 1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}}\rangle$
  Return stp
Alg. wind(stp)
  If $\mathsf{stp} = \perp$ then return $(\perp, \perp)$
  Parse stp as $\langle i, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}}\rangle$
  If $i >$ MW return $(\perp, \perp)$
  $\mathsf{stp}' \leftarrow \langle i+1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}}\rangle$
  Return $(\mathsf{stp}', \mathsf{stm}_i)$
Alg. unwind(stm)
  $\mathsf{stm}' \leftarrow \mathrm{SHA1}(\mathsf{stm})$; Return stm'
Alg. keyder(stm)
  $K \leftarrow \mathrm{SHA1}(0^8 \| \mathsf{stm})$; Return K

The derived key space for the scheme KR-SHA1 is $\mathsf{DK} = \{0,1\}^{160}$. ▮

In practice we assume that the MW might be some reasonable value like 2^{20}. We give a proof of security for KR-SHA1 in Section 9. In our proof of security we view the application of $\mathrm{SHA1}(\cdot)$ in unwind as one random oracle and the application of $\mathrm{SHA1}(0^8\|\cdot)$ in keyder as another random oracle. The proof of security for KR-SHA1 is thus in the random oracle model [9].

	KR-SHA1	KR-AES	KR-RSA
$\mathsf{MW} = \infty$	No	No	Yes
Random oracles	Yes	No	Yes
setup cost	MW SHA1 ops	MW AES ops	1 RSA key generation
wind cost	no crypto	no crypto	1 RSA decryption
unwind cost	1 SHA1 op	1 AES op	1 RSA encryption
keyder cost	1 SHA1 op	1 AES op	1 SHA1 op

Table 1. Our preferred constructions. There are ways of implementing these constructions with different wind costs. The "random oracles" line refers to whether our security proof is in the random oracle model or not.

5.2 The KR-AES construction

Our next preferred construction, KR-AES, uses the AES block cipher and is provably secure in the standard model, meaning without random oracles but assuming that AES is a secure pseudorandom permutation [8, 35].

Construction 5.2 [KR-AES.] The key regression scheme $\mathsf{KR\text{-}AES} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ is defined as follows. MW is a positive integer and a parameter of the construction.

Alg. setup
  $\mathsf{stm}_{\mathsf{MW}} \xleftarrow{\$} \{0,1\}^{128}$
  For $i = \mathsf{MW}$ downto 2 do
    $\mathsf{stm}_{i-1} \leftarrow \mathsf{unwind}(\mathsf{stm}_i)$
  $\mathsf{stp} \leftarrow \langle 1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}} \rangle$
  Return stp
Alg. wind(stp)
  If $\mathsf{stp} = \perp$ then return $(\perp, \perp)$
  Parse stp as $\langle i, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}} \rangle$
  If $i > \mathsf{MW}$ return $(\perp, \perp)$
  $\mathsf{stp}' \leftarrow \langle i+1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}} \rangle$
  Return $(\mathsf{stp}', \mathsf{stm}_i)$
Alg. unwind(stm)
  $\mathsf{stm}' \leftarrow \mathrm{AES}_{\mathsf{stm}}(0^{128})$; Return stm'
Alg. keyder(stm)
  $K \leftarrow \mathrm{AES}_{\mathsf{stm}}(1^{128})$; Return K

The derived key space for the scheme KR-AES is $\mathsf{DK} = \{0,1\}^{128}$. ∎

As with KR-SHA1, we assume that the MW might be some reasonable value like 2^{20}. We prove the security of KR-AES in stages. We first show how to build a secure key regression scheme from any forward-secure pseudorandom bit generator (FSPRG) [10]; we call our construction KR-FSPRG. We then recall one of Bellare and Yee's [10] methods (FSPRG-PRG) for building secure FSPRGs from standard pseudorandom bit generators (PRGs) [10, 11, 51]. Instantiating KR-FSPRG with FSPRG-PRG yields a secure PRG-based key regression scheme that we call KR-PRG. KR-AES is then an instantiation of KR-PRG with a PRG that, on input a 128-bit string stm, outputs $\mathrm{AES}_{\mathsf{stm}}(0^{128}) \| \mathrm{AES}_{\mathsf{stm}}(1^{128})$. Since the constructions KR-FSPRG and KR-PRG have multiple possible instantiations, we consider them to be of independent interest. Details in Sections 6 through 8.

Remark. On can also view KR-SHA1 as an instantiation of KR-PRG with a PRG (in the random oracle model) that, on input a string $\mathsf{stm} \in \{0,1\}^{160}$, outputs $\mathrm{SHA1}(\mathsf{stm}) \| \mathrm{SHA1}(0^8 \| \mathsf{stm})$. In Section 9 we prove KR-SHA1 directly, rather than by instantiating KR-PRG, in order to obtain tighter bounds.

5.3 The KR-RSA construction

Our final preferred construction, KR-RSA derives from the key rotation scheme in Figure 2. KR-RSA differs from KR-SHA1 and KR-AES in that $\mathsf{MW} = \infty$, meaning that a content provider can invoke the KR-RSA winding algorithm an unbounded number of times without violating the correctness properties of key regression schemes. This ability is particularly useful because it means that an implementor need not fix MW to some finite value at implementation or configuration time. Nevertheless, our security proof in Section 10 suggest that in practice a content publisher should limit the number of times it invokes wind to some reasonable value. As another motivation for KR-RSA, we note that if MW is large, then maintaining the publisher states for KR-SHA1 and KR-AES may require a non-trivial amount of space (if the publisher stores the entire vector stp) or time (if the publisher re-derives stp during every wind operation).

Construction 5.3 [KR-RSA.] The key regression scheme $\mathsf{KR\text{-}RSA} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ is defined as follows. Let $\mathcal{K}_{\mathrm{rsa}}$ be an RSA key generator for some security parameter k and let $m\colon \mathbb{Z}_{2^k} \to \{0,1\}^k$ denote the standard big-endian encoding of the integers in $\mathbb{Z}_{2^k}$ to k-bit strings.

Alg. setup
$(N, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$; $S \xleftarrow{\$} \mathbb{Z}_N^*$; $\mathsf{stp} \leftarrow \langle N, e, d, S\rangle$
Return stp
Alg. wind(stp)
Parse stp as $\langle N, e, d, S\rangle$; $S' \leftarrow S^d \bmod N$
$\mathsf{stp}' \leftarrow \langle N, e, d, S'\rangle$; $\mathsf{stm} \leftarrow \langle N, e, S\rangle$
Return $(\mathsf{stp}', \mathsf{stm})$
Alg. unwind(stm)
Parse stm as $\langle N, e, S\rangle$
$S' \leftarrow S^e \bmod N$; $\mathsf{stm}' \leftarrow \langle N, e, S'\rangle$
Return stm'
Alg. keyder(stm)
Parse stm as $\langle N, e, S\rangle$; $K \leftarrow \text{SHA1}(m(S))$
Return K

The derived key space for KR-RSA is $\mathsf{DK} = \{0,1\}^{160}$. In our experiments, we set $k = 1\,024$, and $\mathcal{K}_{\text{rsa}}$ returns $e = 3$ as the RSA public exponent. ▮

The proof of security for KR-RSA is in Section 10. The proof is in the random oracle model and assumes that the RSA key generator is one-way; we define one-wayness in Section 10.

5.4 Discussion

Alternate constructions. Besides KR-SHA1, KR-AES, and KR-RSA, there are numerous possible ways to build key regression schemes, some of which are simple variants of the more general constructions that we present in subsequent sections (KR-FSPRG, KR-PRG, KR-RO, and KR-RSA-RO). Using advanced tree-based schemes [4, 6, 36, 38], a publisher could give access to any contiguous sequence of keys using only a logarithmic number of nodes from a key tree. We do not consider key trees here because one of our primary design goals is to minimize the size of the member states that the content publisher must transmit to members. For instance, it is desirable to have constant-sized metadata in file systems.

On the use of SHA1. We completed the bulk of our research prior to Wang, Yin, and Yu [49] showing how to find collisions in SHA1 faster than brute force. The result of Wang, Yin, and Yu raises the question of whether one should continue to use SHA1 in real constructions, including KR-SHA1 and KR-RSA. This concern is well justified, particularly because other researchers [31, 33] have shown how to extend certain types of collision-finding attacks against hash functions to break cryptosystems that, at first glance, appear to depend only on a weaker property of the underlying hash function (like second-preimage resistance) and therefore initially appear to be immune to collision-finding attacks. Still, we currently suspect that our constructions will resist immediate extensions to collision-finding attacks against SHA1, particularly because the content publisher is the entity responsible for determining the inputs to SHA1 and, under our model, the content publisher would not wish to intentionally compromise the pseudorandomness of its keys. Alternatively, one could replace the use of SHA1 in our constructions with another hash function, perhaps a hash function that behaves like a random oracle assuming that the underlying compression function is a random oracle [15].

6 Key regression from FSPRGs

Toward proving the security of KR-AES, we first show how to construct a key regression scheme from a forward-secure pseudorandom bit generator (FSPRG) [10]. We call our construction KR-FSPRG; see Construction 6.1. Since there are multiple possible ways to instantiate KR-FSPRG, we believe that KR-FSPRG may be of independent interest. Moreover, our result in this section suggests that future work in forward-secure pseudorandom bit generators could have useful applications to key regression schemes.

6.1 Forward-secure pseudorandom generators

Bellare and Yee [10] define stateful pseudorandom bit generators and describe what it means for a stateful pseudorandom bit generator to be forward-secure. Intuitively a stateful PRG is forward-secure if even adversaries that are given the generator's current state cannot distinguish previous outputs from random.

Syntax. A stateful PRG consists of two algorithms: $\mathcal{SBG} = (\mathsf{seed}, \mathsf{next})$. The randomized setup algorithm returns an initial state; we write this as $\mathsf{stg} \xleftarrow{\$} \mathsf{seed}$. The deterministic next step algorithm takes a state as input and returns a new state and an output from $\mathsf{OutSp}_{\mathcal{SBG}}$, or the pair $(\perp, \perp)$; we write this as $(\mathsf{stg}', K) \leftarrow \mathsf{next}(\mathsf{stg})$. We require that the set $\mathsf{OutSp}_{\mathcal{SBG}}$ is efficiently samplable. $\mathsf{MaxLen}_{\mathcal{SBG}} \in \{1, 2, \ldots\} \cup \{\infty\}$ denotes the maximum number of output blocks that $\mathcal{SBG}$ is designed to produce from a correctness (not security) perspective.

Correctness. The correctness requirement for stateful PRGs is as follows: let $\mathsf{stg}_0 \xleftarrow{\$} \mathsf{seed}$ and, for $i = 1, 2, \ldots$, let $(\mathsf{stg}_i, K_i) \xleftarrow{\$} \mathsf{next}(\mathsf{stg}_{i-1})$. We require that for $i \le \mathsf{MaxLen}_{\mathcal{SBG}}$, $(\mathsf{stg}_i, K_i) \ne (\perp, \perp)$.

Security. Let $\mathcal{SBG} = (\mathsf{seed}, \mathsf{next})$ be a stateful bit generator. Let $\mathcal{A}$ be an adversary. Consider the experiments $\mathbf{Exp}^{\text{fsprg-}b}_{\mathcal{SBG},\mathcal{A}}$, $b \in \{0,1\}$, and the oracles NextO_b below. The adversary runs in two stages: find and guess.

Experiment $\mathbf{Exp}^{\text{fsprg-}b}_{\mathcal{SBG},\mathcal{A}}$	**Oracle** NextO_b
$\mathsf{stg} \xleftarrow{\$} \mathsf{seed}$	$(\mathsf{stg}, K) \leftarrow \mathsf{next}(\mathsf{stg})$
$\mathsf{st} \xleftarrow{\$} \mathcal{A}^{\mathsf{NextO}_b}(\mathsf{find})$	If $b = 0$ then
$b' \xleftarrow{\$} \mathcal{A}(\mathsf{guess}, \mathsf{stg}, \mathsf{st})$	$K \xleftarrow{\$} \mathsf{OutSp}_{\mathcal{SBG}}$
Return b'	Return K

The *FSPRG-advantage* of $\mathcal{A}$ in breaking the security of $\mathcal{SBG}$ is defined as

$$\mathbf{Adv}^{\text{fsprg}}_{\mathcal{SBG},\mathcal{A}} = \Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{A}} = 1\right] - \Pr\left[\mathbf{Exp}^{\text{fsprg-0}}_{\mathcal{SBG},\mathcal{A}} = 1\right].$$

Under the concrete security approach, the scheme $\mathcal{SBG}$ is said to be *FSPRG-secure* if the FSPRG-advantage of all adversaries $\mathcal{A}$ using reasonable resources is "small."

6.2 An FSPRG-based key regression scheme

We define KR-FSPRG in Construction 6.1 below. At a high level, one can view KR-FSPRG's setup algorithm as running the FSPRG $\mathcal{SBG}$ backward, meaning setup runs seed and the output of seed becomes KR-FSPRG's MW-th member state. From the MW-th member state, setup invokes next to obtain the $(\mathsf{MW}-1)$-st member state; setup continues in this manner until deriving the 1-st member state. The setup algorithm then outputs a content publisher state stp consisting of an index i, initially 1, and the MW member states. The wind algorithm, on input a publisher state stp with index $i \le \mathsf{MW}$, outputs the i-th member state in the vector and outputs a revised publisher state stp' with index $i+1$. On input a member state stm, the wind and keyder algorithms both invoke next on stm to obtain a pair (stm', K); wind then outputs the revised member state stm' whereas keyder outputs the key K.

Construction 6.1 [KR-FSPRG.] Given a stateful generator $\mathcal{SBG} = (\mathsf{seed}, \mathsf{next})$, we can construct a key regression scheme $\mathsf{KR\text{-}FSPRG} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ as follows. $\mathsf{MW} \le \mathsf{MaxLen}_{\mathcal{SBG}}$ is a positive integer and a parameter of the construction.

Alg. setup
  $\mathsf{stg}_{\mathsf{MW}} \xleftarrow{\$} \mathsf{seed}$
  For $i = \mathsf{MW}$ downto 2 do
    $(\mathsf{stg}_{i-1}, K_{i-1}) \leftarrow \mathsf{next}(\mathsf{stg}_i)$
  $\mathsf{stp} \leftarrow \langle 1, \mathsf{stg}_1, \ldots, \mathsf{stg}_{\mathsf{MW}}\rangle$
  Return stp
Alg. wind(stp)
  If $\mathsf{stp} = \perp$ then return $(\perp, \perp)$
  Parse stp as $\langle i, \mathsf{stg}_1, \ldots, \mathsf{stg}_{\mathsf{MW}}\rangle$
  If $i > \mathsf{MW}$ return $(\perp, \perp)$
  $\mathsf{stp}' \leftarrow \langle i+1, \mathsf{stg}_1, \ldots, \mathsf{stg}_{\mathsf{MW}}\rangle$
  Return $(\mathsf{stp}', \mathsf{stg}_i)$
Alg. unwind(stm)
  $(\mathsf{stm}', K) \leftarrow \mathsf{next}(\mathsf{stm})$; Return stm'
Alg. keyder(stm)
  $(\mathsf{stm}', K) \leftarrow \mathsf{next}(\mathsf{stm})$; Return K

The derived key space for KR-FSPRG is $\mathsf{DK} = \mathsf{OutSp}_{\mathcal{SBG}}$. ∎

In order for setup and wind to be "efficient," we assume that MW has some "reasonable" value like 2^{20}; in the asymptotic setting we would require that MW be polynomial in some security parameter.

Security. The theorem below states that if $\mathcal{SBG}$ is a secure forward-secure pseudorandom bit generator (i.e., is FSPRG-secure), then the resulting key regression scheme KR-FSPRG built from $\mathcal{SBG}$ via Construction 6.1 will be secure (i.e., KR-secure). Specifically, Theorem 6.2 says that given an adversary $\mathcal{A}$ against KR-FSPRG, one can construct an adversary $\mathcal{B}$ against $\mathcal{SBG}$ such that $\mathcal{B}$ uses reasonable resources (if $\mathcal{A}$ does and if MW is small) and Equation (1) in the theorem statement holds; q is the minimum of MW and the maximum number of wind and key derivation oracle queries that $\mathcal{A}$ makes. These properties imply security for KR-FSPRG since, if $\mathcal{SBG}$ is FSPRG-secure and if $\mathcal{A}$ uses reasonable resources, then $\mathbf{Adv}^{\text{fsprg}}_{\mathcal{SBG},\mathcal{B}}$ and q must both be small, which means that $\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{A}}$, the advantage of $\mathcal{A}$ in attacking KR-FSPRG, must be small as well.

Theorem 6.2 *If $\mathcal{SBG}$ is FSPRG-secure, then $\mathcal{KR}$ built from $\mathcal{SBG}$ via* KR-FSPRG *(Construction 6.1) is KR-secure. Concretely, given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{SBG}$ such that*

$$\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{A}} \le (q+1) \cdot \mathbf{Adv}^{\text{fsprg}}_{\mathcal{SBG},\mathcal{B}} \qquad (1)$$

where q is the minimum of MW *and the maximum number of wind and key derivation oracle queries that $\mathcal{A}$ makes. $\mathcal{B}$ makes up to* MW *queries to its oracle and uses within a small constant factor of the other resources of $\mathcal{A}$ plus the time to run the* setup *algorithm.* ∎

Intuitively, Theorem 6.2 follows from the fact that KR-FSPRG runs $\mathcal{SBG}$ backward, which means that if an adversary $\mathcal{A}$ against KR-FSPRG in possession of the first i member states can distinguish a key K_l, $l > i$, from random, then an adversary $\mathcal{B}$ against $\mathcal{SBG}$ in possession of the $(\mathsf{MW}-i)$-th state output of next could distinguish the $(\mathsf{MW}-l)$-th key output of next from random. The actual proof involves $\mathcal{B}$ guessing the number of WindO oracle queries that $\mathcal{A}$ will make. The full proof is in [24].

7 Key regression from standard PRGs

We proceed by showing how to build secure key regression schemes from standard (not forward-secure) pseudorandom bit generators; we call our PRG-based construction KR-PRG. Our approach capitalizes on a method from Bellare and Yee [10] for building FSPRGs from standard PRGs; we recall the Bellare-Yee method in Section 7.1. As with KR-FSPRG from Section 6, we believe that KR-PRG will be of independent interest.

7.1 FSPRGs from pseudorandom bit generators

Pseudorandom bit generators. A pseudorandom bit generator (PRG) [10, 11, 51] is a function $G\colon\ \{0,1\}^k \to \{0,1\}^{k+l}$ that takes as input a k-bit seed and returns a string that is longer than the seed by l bits, $k, l \geq 1$. The standard security notion for a PRG is as follows. If $\mathcal{A}$ is an adversary, we let

$$\begin{aligned}&\mathbf{Adv}^{\mathrm{prg}}_{F,\mathcal{A}}\\ &= \Pr\left[\, K \xleftarrow{\$} \{0,1\}^k\,;\, x \leftarrow G(K) \ :\ \mathcal{A}(x) = 1 \,\right]\\ &\quad -\Pr\left[\, x \xleftarrow{\$} \{0,1\}^{k+l} \ :\ \mathcal{A}(x) = 1 \,\right]\end{aligned}$$

denote the *PRG-advantage* of $\mathcal{A}$ in attacking G. Under the concrete security approach, G is said to be a "secure PRG" if the PRG-advantage of all adversaries $\mathcal{A}$ using reasonable resources is "small."

A PRG-based FSPRG. Bellare and Yee [10] show how to construct an FSPRG from a standard PRG. We dub their scheme FSPRG-PRG and recall it in Construction 7.1 below. The FSPRG-PRG's seed algorithm selects a random k-bit initial seed. The next algorithm, on input a k-bit string stg, computes the $(k+l)$-bit string $G(\mathsf{stg})$ and outputs the first k bits of $G(\mathsf{stg})$ as the next state and the remaining l bits as the key.

Construction 7.1 [FSPRG-PRG, Construction 2.2 of [10].] Given a PRG $G : \{0,1\}^k \to \{0,1\}^{k+l}$ we can construct a FSPRG $\mathcal{SBG} = (\mathsf{seed}, \mathsf{next})$ as shown below

Alg. seed	**Alg.** next(stg_i)
$\mathsf{stg}_0 \xleftarrow{\$} \{0,1\}^k$	$r \xleftarrow{\$} G(\mathsf{stg}_i)$
return stg_0	$\mathsf{stg}_{i+1} \leftarrow$ first k bits of r
	$K \leftarrow$ last l bits of r
	return (stg_{i+1}, K)

The output space of $\mathcal{SBG}$ is $\mathsf{OutSp}_{\mathcal{SBG}} = \{0,1\}^l$ and $\mathsf{MaxLen}_{\mathcal{SBG}} = \infty$. ∎

The following lemma comes from Bellare and Yee [10] except that we treat q as a parameter of the adversary and we allow the trivial case that $q = 0$. Lemma 7.2 states that if G is a secure PRG, then the stateful bit generator FSPRG-PRG built from G via Construction 7.1 will also be secure. Specifically, if G is a secure PRG, then $\mathbf{Adv}^{\mathrm{prg}}_{G,B}$ must be small for all adversaries $\mathcal{B}$ using reasonable resources. Further, if an adversary $\mathcal{A}$ against FSPRG-PRG uses reasonable resources, then the number of oracle queries q that it makes must also be small and $\mathcal{B}$ must also use reasonable resources. These properties, coupled with Equation (2), means that the advantage of all adversaries $\mathcal{A}$ against FSPRG-PRG that use reasonable resources must be small; i.e., FSPRG-PRG must be FSPRG-secure.

Lemma 7.2 ***[Theorem 2.3 of [10].]*** *Let $G : \{0,1\}^k \to \{0,1\}^{k+l}$ be a PRG, and let $\mathcal{SBG}$ be the FSPRG built using G according to Construction 7.1. Given an adversary $\mathcal{A}$ attacking $\mathcal{SBG}$ that makes at most q queries to its oracle, we can construct an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{fsprg}}_{\mathcal{SBG},\mathcal{A}} \leq 2q \cdot \mathbf{Adv}^{\mathrm{prg}}_{G,\mathcal{B}} \qquad (2)$$

where $\mathcal{B}$ uses within a small constant factor of the resources of $\mathcal{A}$ and computes G up to q times. ∎

7.2 A PRG-based key regression scheme

Combining KR-FSPRG and FSPRG-PRG in the natural way yields a key regression scheme that we call KR-PRG. For concreteness we describe KR-PRG in detail below.

Construction 7.3 [KR-PRG.] Let $G\colon\ \{0,1\}^k \to \{0,1\}^{k+l}$ be a pseudorandom bit generator. We can construct a key regression scheme KR-PRG $= (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ from G as follows. MW is a positive integer and a parameter of the construction.

Alg. setup
  $\mathsf{stm}_{\mathsf{MW}} \xleftarrow{\$} \{0,1\}^k$
  For $i = \mathsf{MW}$ downto 2 do
    $\mathsf{stm}_{i-1} \leftarrow \mathsf{unwind}(\mathsf{stm}_i)$
  $\mathsf{stp} \leftarrow \langle 1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}}\rangle$
  Return stp
Alg. wind(stp)
  If $\mathsf{stp} = \perp$ then return $(\perp, \perp)$
  Parse stp as $\langle i, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}}\rangle$
  If $i > \mathsf{MW}$ return $(\perp, \perp)$
  $\mathsf{stp}' \leftarrow \langle i+1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}}\rangle$
  Return $(\mathsf{stp}', \mathsf{stm}_i)$
Alg. unwind(stm)
  $x \leftarrow G(\mathsf{stm})\,;\ \mathsf{stm}' \leftarrow$ first k bits of x
  Return stm′
Alg. keyder(stm)
  $x \leftarrow G(\mathsf{stm})\,;\ K \leftarrow$ last l bits of x
  Return K

The derived key space for KR-PRG is $\mathsf{DK} = \{0,1\}^l$. ∎

In order for setup and wind to be "efficient," we assume that MW has some "reasonable" value like 2^{20}; in the asymptotic setting we would require that MW be polynomial in some security parameter.

Security. The theorem below states that if G is a secure PRG, then the resulting key regression scheme KR-PRG built from G via Construction 7.3 will be KR-secure. Specifically, Theorem 7.4 says that given an adversary $\mathcal{A}$ against KR-PRG that uses reasonable resources, and assuming that MW is small, one can construct an adversary $\mathcal{B}$ against G such that $\mathcal{B}$ uses reasonable resources and Equation (3) in the theorem statement holds; q is the minimum of MW and the maximum number of wind and key derivation oracle queries that $\mathcal{A}$ makes. These properties imply security for KR-PRG since, if G is PRG-secure and since $\mathcal{A}$ uses reasonable resources, $\mathbf{Adv}^{\mathrm{prg}}_{G,\mathcal{B}}$ and q must both be small, which means that $\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}}$, the advantage of $\mathcal{A}$ in attacking KR-PRG, must be small as well.

Theorem 7.4 *If $G\colon \{0,1\}^k \to \{0,1\}^{k+l}$ is a secure PRG, then the key regression scheme $\mathcal{KR}$ built from G via* KR-PRG *(Construction 7.3) is KR-secure. Concretely, given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$, we can construct an adversary $\mathcal{B}$ attacking G such that*

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} \;\le\; 2\cdot(q+1)^2 \cdot \mathbf{Adv}^{\mathrm{prg}}_{G,\mathcal{B}} \qquad (3)$$

where q is the minimum of MW *and the maximum number of queries $\mathcal{A}$ makes to its* WindO *and* KeyderO *oracles. Adversary $\mathcal{B}$ uses within a small constant factor of the resources of $\mathcal{A}$, plus the time to compute* setup *and G* MW *times.* ∎

Proof of Theorem 7.4: Construction 7.3 is exactly Construction 6.1 built from the forward secure pseudorandom bit generator defined by Construction 7.1. The theorem statement therefore follows from Theorem 6.2 and Lemma 7.2. ∎

8 The security of KR-AES

Having shown how to construct secure key regression schemes from secure pseudorandom bit generators (KR-PRG and Construction 7.3), we are now able to prove the security of KR-AES (Construction 5.2) by observing that KR-AES is exactly KR-PRG with $k = l = 128$ and with the PRG G defined as $G(X) = \mathrm{AES}_X(0^{128}) \| \mathrm{AES}_X(1^{128})$ for all $X \in \{0,1\}^{128}$. Before stating our formal result for KR-AES, we first recall the standard notion of a pseudorandom permutation [8, 35].

Pseudorandom permutations. Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher and let $\mathrm{Perm}(l)$ denote the set of all permutations on $\{0,1\}^l$. If $\mathcal{A}$ is an adversary with access to an oracle, we let

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prp}}_{E,\mathcal{A}} &= \Pr\left[\, K \xleftarrow{\$} \{0,1\}^k \;:\; \mathcal{A}^{E_K(\cdot)} = 1 \,\right] \\ &\quad -\Pr\left[\, g \xleftarrow{\$} \mathrm{Perm}(l) \;:\; \mathcal{A}^{g(\cdot)} = 1 \,\right]\end{aligned}$$

denote the *PRP-advantage* of $\mathcal{A}$ in attacking E. Under the concrete security approach, E is said to be a "secure PRP" if the PRP-advantage of all adversaries $\mathcal{A}$ using reasonable resources is "small."

Instantiating KR-AES **from** KR-PRG**.** As noted above, it is straightforward to instantiate KR-AES from KR-PRG. Numerous other instantiations exist, e.g., to use a block cipher E with $k > l$, one might define G as $G(X) = E_X(\alpha_1) \| E_X(\alpha_2) \| \ldots$ where $\alpha_1, \alpha_2, \ldots$ are distinct l-bit strings. Since KR-AES is one of our preferred constructions, we state the following theorem specifically for KR-AES; it is straightforward to extend our result to other natural instantiations of KR-PRG. The security proof for KR-AES is in the standard model and assumes that AES is a secure pseudorandom permutation.

Theorem 8.1 *If AES is a secure PRP, then* KR-AES *(Construction 5.2) is KR-secure. Concretely, given an adversary $\mathcal{A}$ attacking* KR-AES*, we can construct an adversary $\mathcal{B}$ attacking AES such that*

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} \;\le\; 2\cdot(q+1)^2 \cdot \left(\mathbf{Adv}^{\mathrm{prp}}_{AES,\mathcal{B}} + 2^{-128}\right) \quad (4)$$

where q is the minimum of MW *and the maximum number of queries $\mathcal{A}$ makes to its* WindO *and* KeyderO *oracles. Adversary $\mathcal{B}$ makes 2 oracle queries and uses within a small constant factor of the resources of $\mathcal{A}$, plus the time to compute* setup *and AES* 2MW *times.* ∎

We interpret Theorem 8.1 as follows. Suppose $\mathcal{A}$ is an adversary against KR-AES that uses reasonable resources, and in particular makes at most a reasonable number of queries q to its wind and key derivation oracles. Then we can construct an adversary $\mathcal{B}$ against AES that also uses reasonable resources when MW is small. Because of the resource restrictions on $\mathcal{B}$ and under the assumption that AES is a secure PRP, it follows that $\mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES},\mathcal{B}}$ must be small. If both q and $\mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES},\mathcal{B}}$ are small, then by Equation (4) $\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}}$ must also be small, meaning that KR-AES must be KR-secure.

As a concrete example of the bound in Theorem 8.1, consider the case where MW and q are both 2^{20}. Then Equation (4) becomes

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} \;\le\; 2^{42} \cdot \mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES},\mathcal{B}} + 2^{-86}\,,$$

which means that unless $\mathcal{A}$ exploits a property of AES itself, $\mathcal{A}$ will not be able to break the security of KR-AES with probability better than 2^{-86}. Since it is widely believed that AES is secure, Theorem 8.1 tells us that it is reasonable to assume that KR-AES is secure for reasonable choices of MW.

To prove Theorem 8.1 we use Theorem 7.4, the relationship between KR-AES and KR-PRG, and the fact that the function G defined as $G(X) = \text{AES}_X(0^{128}) \| \text{AES}_X(1^{128})$, $X \in \{0,1\}^{128}$, is a secure PRG if AES is a secure PRP. Details in [24].

9 The security of KR-SHA1

Although we derived KR-SHA1 from the key rotation scheme in Figure 3, we find that one can also view KR-SHA1 as an instantiation of KR-PRG with $k = l = 160$ and G defined as $G(X) = \text{SHA1}(X) \| \text{SHA1}(0^8 \| X)$ for all $X \in \{0,1\}^{160}$. If we view SHA1 as a random oracle, then G is a secure PRG in the random oracle model, and we can use this observation and Theorem 7.4 to prove the security of KR-SHA1 in the random oracle model.

Here we give a direct proof of security for KR-SHA1 in order to obtain a tighter bound. The tightness issue with using KR-PRG and Theorem 7.4 to prove the security of KR-SHA1 rests in the fact that the advantage of an adversary in attacking G in the random oracle model must be upper bounded by a function of the number of random oracle queries that the adversary makes, and this function will percolate through the bound in Theorem 7.4.

In what follows we view $\text{SHA1}(\cdot)$ in KR-SHA1's unwind algorithm and $\text{SHA1}(0^8 \| \cdot)$ in KR-SHA1's keyder algorithm as two different random oracles. Construction 9.1, KR-RO, makes this generalization of KR-SHA1 concrete. We choose not to model $\text{SHA1}(\cdot)$ and $\text{SHA1}(0^8 \| \cdot)$ as a single random oracle because we do not wish to restrict our analysis to the case where keyder must prefix its inputs to the random oracle with the zero byte.

Construction 9.1 [KR-RO.] Let $H_1\colon \{0,1\}^k \to \{0,1\}^k$ and $H_2\colon \{0,1\}^k \to \{0,1\}^l$ be random oracles. We can construct a key regression scheme KR-RO $= (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ from H_1 and H_2 as shown below. MW is a positive integer and a parameter of the construction.

Alg. setup^{H_1,H_2}
  $\mathsf{stm}_{\mathsf{MW}} \xleftarrow{\$} \{0,1\}^k$
  For $i = \mathsf{MW}$ downto 2 do
    $\mathsf{stm}_{i-1} \leftarrow \mathsf{unwind}^{H_1,H_2}(\mathsf{stm}_i)$
  $\mathsf{stp} \leftarrow \langle 1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}} \rangle$
  Return stp

Alg. $\mathsf{wind}^{H_1,H_2}(\mathsf{stp})$
  If $\mathsf{stp} = \perp$ then return $(\perp, \perp)$
  Parse stp as $\langle i, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}} \rangle$
  If $i > \mathsf{MW}$ return $(\perp, \perp)$
  $\mathsf{stp}' \leftarrow \langle i+1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}} \rangle$
  Return $(\mathsf{stp}', \mathsf{stm}_i)$

Alg. $\mathsf{unwind}^{H_1,H_2}(\mathsf{stm})$
  $\mathsf{stm}' \leftarrow H_1(\mathsf{stm})$; Return stm'

Alg. $\mathsf{keyder}^{H_1,H_2}(\mathsf{stm})$
  $K \leftarrow H_2(\mathsf{stm})$; Return K

The derived key space for KR-RO is $\mathsf{DK} = \{0,1\}^l$. ∎

In order for setup and wind to be "efficient," we assume that MW has some "reasonable" value like 2^{20}; in the asymptotic setting we would require that MW be polynomial in some security parameter.

The following theorem states that Construction 9.1 is secure in the random oracle model for adversaries that make a reasonable number of queries to their random oracles.

Theorem 9.2 *The key regression scheme in Construction 9.1 is secure in the random oracle model. Formally, let $H_1\colon \{0,1\}^k \to \{0,1\}^k$ and $H_2\colon \{0,1\}^k \to \{0,1\}^l$ be random oracles and let $\mathcal{KR}$ be the key regression scheme built from H_1, H_2 via* KR-RO *(Construction 9.1). Then for any adversary $\mathcal{A}$ we have that*

$$\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{A}} \;\le\; \frac{(\mathsf{MW})^2}{2^{k+1}} + \frac{q \cdot \mathsf{MW}}{2^k - \mathsf{MW} - q}\,, \qquad (5)$$

where q is the maximum number of queries total that $\mathcal{A}$ makes to its H_1 and H_2 random oracles. ∎

As a concrete example of the bound in Theorem 9.2, consider the case where $\mathsf{MW} = 2^{20}$ and an adversary $\mathcal{A}$ makes at most $q = 2^{40}$ queries to its random oracles. Then Equation (5) tells us that the advantage of $\mathcal{A}$ in attacking KR-RO is upper bounded by 2^{-98}. Although SHA1 is not a random oracle, Theorem 9.2 gives us confidence that KR-SHA1 may provide a reasonable level of security in practice; see Section 5 for additional discussion.

We prove Theorem 9.2 in [24], but remark that we could simplify the proof if, instead of defining KR-RO as in Construction 9.1, we include the indices i in the member states, and hence in the inputs to H_1 and H_2. We choose to omit the indices i from the member states in KR-RO because we view KR-RO and KR-SHA1 as closer to what developers might wish to implement in practice.

We remark that in addition to viewing KR-SHA1 as an instantiation of KR-PRG, one could view KR-AES as an instantiation of KR-RO with $k = l = 128$ and, for all $X \in \{0,1\}^{128}$, with $H_1(X)$ defined as $\text{AES}_X(0^{128})$

and $H_2(X)$ defined as $\text{AES}_X(1^{128})$; Diffie and Hellman suggest using a block cipher as a hash function in this manner in [16]. We choose to prove the security of KR-AES directly in Section 8, rather than instantiate KR-RO, because we desire a proof of security for KR-AES in the standard model.

10 The security of KR-RSA

In our proof of security for KR-RSA we view the use of SHA1 in keyder as a random oracle. Construction 10.1, KR-RSA-RO, makes this generalization concrete.

Construction 10.1 [KR-RSA-RO.] Given an RSA key generator $\mathcal{K}_{\text{rsa}}$ for some security parameter k and a random oracle $H\colon \mathbb{Z}_{2^k} \to \{0,1\}^l$, we can construct a key regression scheme $\mathsf{KR\text{-}RSA\text{-}RO} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ as shown below, where $\mathsf{MW} = \infty$.

Alg. setup^H
  $(N, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$; $S \xleftarrow{\$} \mathbb{Z}_N^*$; $\mathsf{stp} \leftarrow \langle N, e, d, S\rangle$
  Return stp
Alg. $\mathsf{wind}^H(\mathsf{stp})$
  Parse stp as $\langle N, e, d, S\rangle$; $S' \leftarrow S^d \bmod N$
  $\mathsf{stp}' \leftarrow \langle N, e, d, S'\rangle$; $\mathsf{stm} \leftarrow \langle N, e, S\rangle$
  Return $(\mathsf{stp}', \mathsf{stm})$
Alg. $\mathsf{unwind}^H(\mathsf{stm})$
  Parse stm as $\langle N, e, S\rangle$
  $S' \leftarrow S^e \bmod N$; $\mathsf{stm}' \leftarrow \langle N, e, S'\rangle$
  Return stm'
Alg. $\mathsf{keyder}^H(\mathsf{stm})$
  Parse stm as $\langle N, e, S\rangle$; $K \leftarrow H(S)$
  Return K

The derived key space for the scheme KR-RSA-RO is $\mathsf{DK} = \{0,1\}^l$. ∎

Toward proving KR-RSA secure, we first prove in Section 10.1 that KR-RSA-RO is KR-secure against adversaries that use reasonable resources and that make at most one KeyderO oracle query; the result in Section 10.1 assumes that the RSA key generator $\mathcal{K}_{\text{rsa}}$ in KR-RSA-RO is one-way. We then show in Section 10.2 that if a key regression scheme is secure against adversaries restricted to one KeyderO oracle query, then the key regression scheme is secure against adversaries making multiple KeyderO oracle queries. In Section 10.3 we combine these two results to show that KR-RSA-RO is secure against adversaries that use reasonable resources but make an otherwise unrestricted number of KeyderO oracle queries.

Before proceedings with Section 10.1, we first define what it means for an RSA key generator to be one-way.

Security for RSA key generators. Let $\mathcal{K}_{\text{rsa}}$ be an RSA key generator with security parameter k. If $\mathcal{A}$ is an adversary, we let

$$\mathbf{Adv}^{\text{rsa-ow}}_{\mathcal{K}_{\text{rsa}},\mathcal{A}} = \Pr\left[\begin{array}{c|c} (N,e,d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}\,; & \\ x \xleftarrow{\$} \mathbb{Z}_N^*\,; & \mathcal{A}(y,e,N) = x \\ y \leftarrow x^e \bmod N & \end{array}\right]$$

denote the RSA one-way advantage of $\mathcal{A}$ in inverting RSA with the key generator $\mathcal{K}_{\text{rsa}}$. Under the concrete security approach, $\mathcal{K}_{\text{rsa}}$ is said to be a "one-way" if the RSA one-way advantage of all adversaries $\mathcal{A}$ using reasonable resources is "small."

10.1 Security of KR-RSA under one KeyderO oracle query

Lemma 10.2 below states that if the RSA key generator $\mathcal{K}_{\text{rsa}}$ is one-way, then the resulting construction KR-RSA-RO is secure against adversaries that use reasonable resources and that make at most one KeyderO oracle query.

Lemma 10.2 *If $\mathcal{K}_{rsa}$ is an RSA key generator with security parameter k, then the key regression scheme $\mathcal{KR}$ built from $\mathcal{K}_{rsa}$ via* KR-RSA-RO *(Construction 10.1) is KR-secure in the random oracle model against adversaries restricted to one* KeyderO *oracle query assuming that $\mathcal{K}_{rsa}$ is one-way. Concretely, given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$ that makes at most one key derivation oracle query, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{K}_{rsa}$ such that*

$$\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{A}} \le (q+1) \cdot \mathbf{Adv}^{\text{rsa-ow}}_{\mathcal{K}_{rsa},\mathcal{B}}\ , \qquad (6)$$

where q is the maximum number of winding oracle queries that $\mathcal{A}$ makes. Adversary $\mathcal{B}$ uses within a small constant factor of the resources as $\mathcal{A}$ plus performs up to q RSA encryption operations. ∎

To prove Lemma 10.2 we observe that in order for an adversary $\mathcal{A}$ in possession of the i-th member state $\langle N, e, S_i\rangle$ to distinguish the $(i+1)$-st key from random, the adversary must query its random oracle with S_{i+1}, where $\langle N, e, S_{i+1}\rangle$ is the $(i+1)$-st member state. Since $S_i = S_{i+1}^e \bmod N$, querying the random oracle with S_{i+1} amounts to inverting S_i. The actual proof of Lemma 10.2 involves $\mathcal{B}$ guessing the number of WindO oracle queries that $\mathcal{A}$ makes. Details in [24].

10.2 Security under one KeyderO oracle query implies security under many

The following lemma states that if a key regression scheme is secure against adversaries restricted to one

KeyderO oracle query, then the key regression scheme is secure against adversaries allowed multiple KeyderO oracle queries. The proof of Lemma 10.3 is in [24].

Lemma 10.3 *If a key regression scheme is secure when an adversary is limited to one* KeyderO *oracle query, then the key regression scheme is secure when an adversary is allowed multiple* KeyderO *oracle queries. Concretely, let $\mathcal{KR}$ be a key regression scheme. Given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$ that makes at most q_1 queries to* WindO *and q_2 queries to* KeyderO*, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{KR}$ such that*

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} \;\leq\; q_2 \cdot \mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{B}}\;, \qquad (7)$$

$\mathcal{B}$ makes at most $q_1 + q_2 - 1$ queries to WindO *(or 0 queries if $q_1 + q_2 = 0$), $\mathcal{B}$ makes at most one query to* KeyderO*, and $\mathcal{B}$ has other resource requirements within a small constant factor of the resource requirements of $\mathcal{A}$.* ∎

10.3 The security of KR-RSA under multiple KeyderO oracle queries

From Lemma 10.2 and Lemma 10.3 it follows that KR-RSA-RO is secure in the random oracle model assuming that $\mathcal{K}_{\mathrm{rsa}}$ is one-way, even for adversaries allowed multiple KeyderO oracle queries. Theorem 10.4 makes this reasoning formal. Although SHA1 is not a random oracle, Theorem 10.4 suggests that when instantiated with a suitable RSA key generator, KR-RSA may provide a reasonable level of security in practice; see Section 5 for additional discussion.

Theorem 10.4 *If $\mathcal{K}_{rsa}$ is an RSA key generator with security parameter k, then $\mathcal{KR}$ built from $\mathcal{K}_{rsa}$ via* KR-RSA-RO *(Construction 10.1) is KR-secure in the random oracle model under the RSA assumption. Concretely, given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{K}_{rsa}$ such that*

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} \leq 2q^2 \cdot \mathbf{Adv}^{\text{rsa-ow}}_{\mathcal{K}_{rsa},\mathcal{B}}\;,$$

where q is the maximum number of winding and key derivation oracle queries that $\mathcal{A}$ makes. Adversary $\mathcal{B}$ uses resources within a constant factor of the resources of $\mathcal{A}$ plus the time to perform q RSA encryption operations. ∎

Proof of Theorem 10.4: The proof of Theorem 10.4 follows from Lemma 10.3 and Lemma 10.2. Note that for the application of Lemma 10.3 we set $q_1 = q$ and $q_2 = q$, meaning the adversary $\mathcal{B}$ from Lemma 10.3 may make up to $2q - 1$ queries to its WindO oracle, or $2q$ if $q = 0$. ∎

11 Performance of key regression in access-controlled content distribution

We integrated key regression into the Chefs file system [22] to measure the performance characteristics of key regression in a real application. We first give an overview of Chefs. Then we provide measurements to show that key regression enables efficient key distribution even for publishers with low-bandwidth and high-latency connections such as cable and analog modems.

Chefs for access-controlled content distribution. Chefs [22] is a secure, single-writer, many-reader file system for access-controlled content distribution using untrusted servers. Chefs extends the SFS read-only file system [23] to provide access control. Chefs uses lazy revocation [21, 32] and KR-SHA1 key regression to reduce the amount of out-of-band communication necessary for group key distribution.

Three modules comprise the Chefs file system. An *untrusted server* makes encrypted, integrity-protected content available in the form of a block store. A *publisher* creates the encrypted, integrity-protected content and manages key distribution. A *client* downloads content from an untrusted server, then verifies integrity and decrypts the content using keys fetched from the publisher. Our publisher, e.g., a blogger, is expected to have a low-bandwidth connection.

Several types of keys guard the access control and confidentiality of content in Chefs. Chefs uses a *content key* to encrypt content. A member obtains a content key by opening a lockbox that is encrypted with the *group key*; the member derives the group key from the group member state. After a membership event, e.g., an eviction, the publisher produces a new key regression member state. The remaining group members request this member state on-demand from the publisher; to communicate the new member state, the publisher encrypts the member state with each member's 1 024-bit public RSA key using the low exponent $e = 3$.

11.1 Hypothesis and methodology

Performance measurements validate that (1) key regression allows a publisher to serve many keys per second to clients effectively independent of the publisher's network throughput and the rate of membership turnover, and (2) key regression does not degrade client latency. To test these hypotheses, we compare the performance of Chefs to Sous-Chefs, a version of Chefs without key regression.

Experimental setup. The client and server contained the same hardware: a 2.8 GHz Intel Pentium 4 with 512 MB RAM. Each machine used a 100 Mbit/sec

full-duplex Intel PRO/1000 Ethernet card and a Maxtor 250 GB, Serial ATA 7 200 RPM hard drive with an 8 MB buffer size, 150 MB/sec transfer rate, and less than 9.0 msec average seek time. The publisher was a 3.06 GHz Intel Xeon with 2 GB RAM, a Broadcom BCM5704C Dual Gigabit Ethernet card, and a Hitachi 320 GB SCSI-3 hard drive with a 320 MB/sec transfer rate.

The machines were connected on a 100 Mbit/sec local area network and all used FreeBSD 4.9. NetPipe [46] measured the round-trip latency between the pairs of machines at 249 μsec, and the maximum sustained TCP throughput of the connection at 88 Mbit/sec when writing data in 4 MB chunks and using TCP send and receive buffers of size 69 632 KB. When writing in 8 KB chunks (the block size in Chefs), the peak TCP throughput was 66 Mbit/sec.

The experiments used the dummynet [44] driver in FreeBSD to simulate cable modem and analog modem network conditions. For the cable modem on the publisher machine, the round-trip delay was set to 20 msec and the download and upload bandwidth to 4 Mbit/sec and 384 Kbit/sec respectively. For the analog modem, the round-trip delay was set to 200 msec and the upload and download bandwidth each to 56 Kbit/sec.

In the Chefs measurements, the inode cache has 16 384 entries, a directory block cache has 512 entries, an indirect block cache has 512 entries, and a file block cache has 64 entries. A large file block cache is unnecessary because the NFS loopback server performs most of the file data caching.

For each measurement, the median result of five samples are reported. Errors bars in Figure 5 indicate minimum and maximum measurements.

11.2 Secure content distribution on untrusted storage

A standard benchmark is not available for measuring the effects of group membership dynamics. Therefore, we evaluate Chefs based on how a client might search for content in a subscription-based newspaper.

Table 2 displays the performance of basic key regression operations. The internal block size of the hash function matters significantly for the throughput of KR-SHA1 key regression. Because SHA1 uses an internal 512-bit block size, hashing values smaller than 512 bits results in poorer throughput than one would expect from SHA1 hashing longer inputs. For this reason, KR-AES key regression performs significantly better than KR-SHA1 key regression.

Searching encrypted content. The benchmarks were inspired by the membership dynamics reported at Salon.com, a subscription-based online journal [45]. Salon announced that in the year 2003, they added 31 000 paid subscribers (for a total of 73 000) and maintained a 71% renewal rate. Thus, a 29% eviction rate would generate an expected 21 170 evictions in one year. This suggests that the total number of membership events would reach 52 170.

To represent a workload of searching newspaper content, the experiment tests a file system containing 10 000 8 KB encrypted files and the associated content keys. The experiment consists of mounting the file system and reading all the files. This causes the client machine to fetch all the content keys.

We further motivate our example workload as follows. While there is promising research in enabling a third party server to search encrypted data [2, 12, 26, 28, 47, 50], current approaches for searchable encryption do not prevent the server from outputting false negatives. Because Chefs extends the SFS read-only file system, it inherits the property that the client can verify whether it has received all intended content (i.e., the whole truth) from the server. Therefore, to avoid false negatives, we desire a client-based search in which the Chefs client downloads all the encrypted content and keys to perform the search itself.

Sous-Chefs. To determine the cost of key regression, Chefs is compared to a version of Chefs with key regression disabled. This strawman file system is called Sous-Chefs. Chefs and Sous-Chefs differ only in how they fetch group keys from the publisher. When using KR-SHA1 for key regression, Chefs fetches a 20-byte member state, encrypted in the client's public 1 024-bit RSA key with low exponent $e = 3$. Chefs then uses key regression to unwind and derive all past versions of the group key. Sous-Chefs fetches all the derived group keys at once (each 16 bytes). The group keys themselves are encrypted with 128-bit AES in CBC mode. The AES key is encrypted with the client's RSA public key. A Sous-Chefs client is allowed to request a single bulk transfer of every version of a group key to fairly amortize the cost of the transfer.

Reduced throughput requirements. Figure 4 shows that a publisher can serve many more clients in Chefs than Sous-Chefs in low-bandwidth, high-latency conditions. The CPU utilization for Chefs under no bandwidth limitation is negligible, indicating that the cost of RSA encryptions on the publisher is not the bottleneck.

Each test asynchronously plays back 20 traces of a single client fetching the keys for the search workload. This effectively simulates the effect of 20 clients applying the same key distribution workload to the publisher. After all traces have completed, we record the effective number of trace playbacks per second. The Sous-Chefs traces of fetching 10, 10^2, 10^3, 10^4, 10^5, and 10^6 keys generate 4, 4, 5, 24, 200, and 1 966 asynchronous remote

Key regression protocol	Winds/sec	Unwinds/sec
KR-SHA1	Not applicable	687 720
KR-AES	Not applicable	3 303 900
KR-RSA	158	35 236

Table 2. Microbenchmarks of KR-SHA1, KR-AES, KR-RSA key regression.

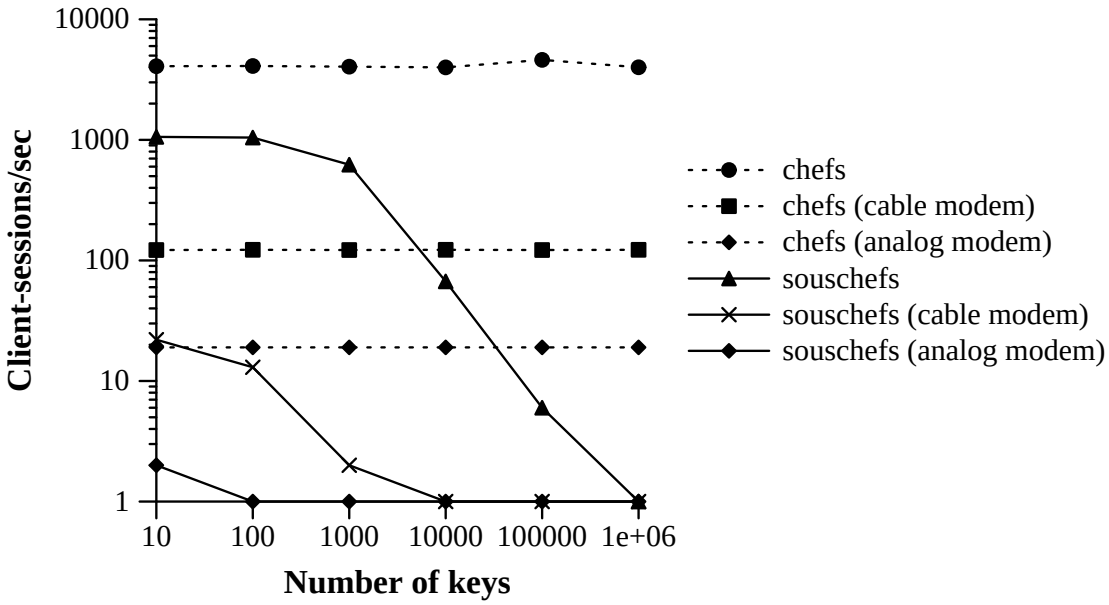

Figure 4. Aggregate publisher throughput for key distribution plotted on a log-log graph. A client-session consists of fetching key material sufficient to generate all the keys to decrypt the published content. Key regression enables a publisher to support many client-sessions per second. Chefs always performs better than Sous-Chefs because key regression performance is effectively independent of the rate of membership turnover.

procedure calls from the client to the publisher respectively. Chefs always generates a single remote procedure call, regardless of the number of key versions.

Improved client latency. The client latency experiment measures the time for a single client to execute the search workload. The untrusted server and publisher have warm caches while the client has a cold cache.

The log-log chart in Figure 5 shows that Chefs outperforms Sous-Chefs for the search workload under several network conditions. In Sous-Chefs, the network transfer time dominates client latency because of the sheer volume of keys transferred from the publisher to the client. There is no measurement for Sous-Chefs downloading 1 000 000 keys because the kernel assumes that the mount failed after waiting 1 000 seconds. On a 56 Kbit/sec network, Sous-Chefs is expected to take over 2 232 seconds to download 1 000 000 keys each 16 bytes. Thus, only three bars appear for the test cases involving 1 000 000 content keys. Key regression itself is a small component of the Chefs benchmark. With 10^6 keys, key regression on the client takes less than 1.5 sec with CPU utilization never exceeding of 42%.

12 Conclusions

We presented provably-secure constructions for key regression — addressing the shortfalls of key rotation. We also provided the first measurements of either a key regression or key rotation system. Finally, we integrated key regression in a content distribution application to demonstrate how key regression enables efficient key distribution on low-bandwidth, high-latency connections. Using key regression, a publisher can efficiently control access to content independent of group membership dynamics and without needing a fast network connection.

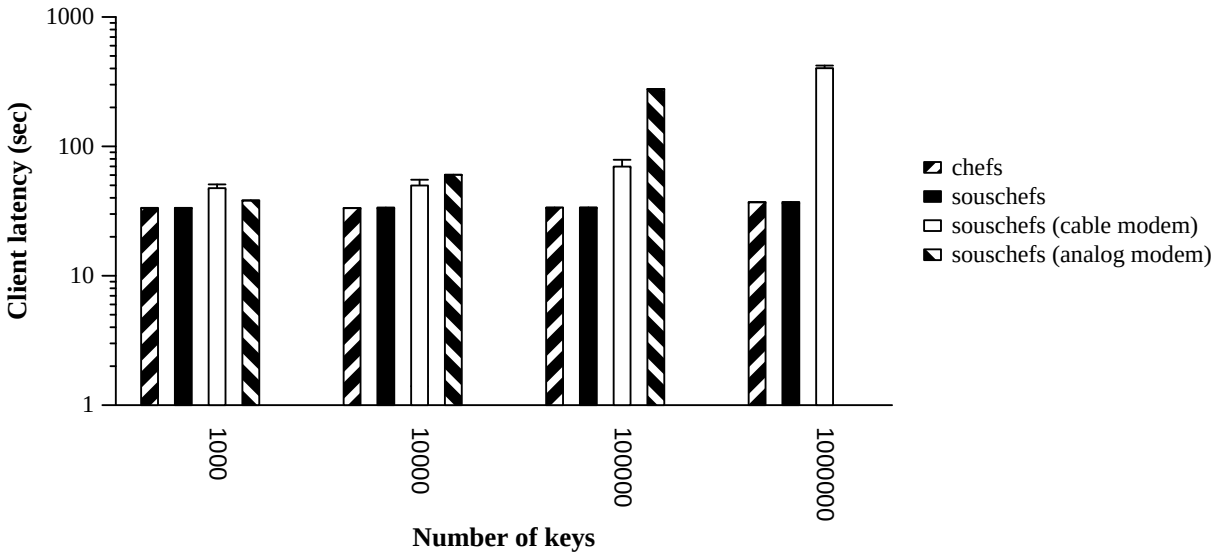

Figure 5. A log-log chart of single client latency to read 10 000 8 KB encrypted files and the associated content keys. Key regression maintains a constant client latency regardless of the number of keys. Under low-bandwidth, high-latency conditions, Sous-Chefs latency is dominated my the transfer time of keys after reaching 10 000 keys. Key regression enables much better latency in Chefs.

Acknowledgments

K. Fu was supported in part by Project Oxygen and an Intel Ph.D. Fellowship. S. Kamara was supported by a Bell Labs Graduate Research Fellowship. T. Kohno was supported by an IBM Ph.D. Fellowship, NSF CCR-0208842, NSF ANR-0129617, and NSF CCR-0093337. K. Fu performed this research while at The Johns Hopkins University and MIT. T. Kohno performed part of this research while visiting UC Berkeley. We thank Ron Rivest for detailed comments on this paper; David Mazières for suggestions on formalizing definitions of security; Mahesh Kallahalla and Ram Swaminathan for our initial work together to define key regression; Fabian Monrose for early reviews of this paper; Frans Kaashoek for his guidance and unending support; and Frank Dabek, Emil Sit, and Jeremy Stribling for help with the testbed.

References

[1] M. Abdalla and M. Bellare. Increasing the lifetime of a key: A comparitive analysis of the security of re-keying techniques. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 546–559, Kyoto, Japan, Dec. 3–7, 2000.

[2] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, Santa Barbara, CA, USA, Aug. 14–18, 2005. Springer-Verlag, Berlin, Germany.

[3] *Akamai Technologies*. `http://www.akamai.com`.

[4] S. G. Akl and P. D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, 1983.

[5] M. Backes, C. Cachin, and A. Oprea. Lazy revocation in cryptographic file systems. In *3rd International IEEE Security in Storage Workshop*, Dec. 2005.

[6] M. Backes, C. Cachin, and A. Oprea. Secure key-updating for lazy revocation. IBM Research Report RZ 3627, Oct. 2005. Available at `http://domino.research.ibm.com/library/cyberdig.nsf/index.html`, keyword 99637; also archived as Cryptology ePrint Archive Report 2005/334.

[7] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 394–403. IEEE Computer Society, 1997.

[8] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358, Santa Barbara, CA, USA, Aug. 21–25, 1994. Springer-Verlag, Berlin, Germany.

[9] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, Lecture Notes in Computer Science, Fairfax, Virginia, USA, Nov. 3–5, 1993. ACM Press.

[10] M. Bellare and B. Yee. Forward security in private key cryptography. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, Apr. 13–17, 2003. Springer-Verlag, Berlin, Germany.

[11] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, 1982.

[12] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522, Interlaken, Switzerland, May 2–6, 2004. Springer-Verlag, Berlin, Germany.

[13] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275, Santa Barbara, CA, USA, Aug. 14–18, 2005. Springer-Verlag, Berlin, Germany.

[14] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[15] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.

[16] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1978.

[17] Y. Dodis and N. Fazio. Public key broadcast encryption for stateless receivers. In *Digital Rights Management Workshop*, volume 2696 of *Lecture Notes in Computer Science*, pages 61–80. Springer-Verlag, Berlin, Germany, 2002.

[18] Y. Dodis and N. Fazio. Public key broadcast encryption secure against adaptive chosen ciphertext attack. In Y. Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 100–115, Miami, USA, Jan. 6–8, 2003. Springer-Verlag, Berlin, Germany.

[19] A. Fiat and M. Naor. Broadcast encryption. In D. Boneh, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 22–26, Santa Barbara, CA, USA, Aug. 17–21, 1993. Springer-Verlag, Berlin, Germany.

[20] M. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, San Francisco, CA, March 2004. See `http://www.coralcdn.org/`.

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

[22] K. Fu. *Integrity and access control in untrusted content distribution networks*. PhD thesis, Massachusetts Institute of Technology, September 2005.

[23] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. In *4th Symposium on Operating Systems Design and Implementation*, 2000.

[24] K. Fu, S. Kamara, and T. Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. Cryptology ePrint Archive `http://eprint.iacr.org/`: Report 2005/303, 2005. (Full version of this paper.).

[25] D. K. Gifford. Cryptographic sealing for information secrecy and authentication. *Communications of the ACM*, 25(4):274–286, 1982.

[26] E.-J. Goh. Secure indexes. Cryptology ePrint Archive `http://eprint.iacr.org/`: Report 2003/216, 2003.

[27] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, Apr. 1984.

[28] P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In M. Jakobsson, M. Yung, and J. Zhou, editors, *ACNS 04: 2nd International Conference on Applied Cryptography and Network Security*, volume 3089 of *Lecture Notes in Computer Science*, pages 31–45, Yellow Mountain, China, June 8–11, 2004. Springer-Verlag, Berlin, Germany.

[29] N. M. Haller. The S/KEY one-time password system. In *ISOC Symposium on Network and Distributed System Security*, February 1994.

[30] A. Harrington and C. Jensen. Cryptographic access control in a distributed file system. In *Proceedings of 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, Villa Gallia, Como, Italy, June 2003.

[31] A. Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer-Verlag, Berlin, Germany, 2004.

[32] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *2nd USENIX Conference on File and Storage Technologies*, 2003.

[33] J. Kelsey and T. Kohno. Herding hash functions and the Nostradamus attack. Cryptology ePrint Archive `http://eprint.iacr.org/`: Report 2005/281, 2005.

[34] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–771, November 1981.

[35] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2), 1988.

[36] S. MacKinnon and S. G. Akl. New key generation algorithms for multilevel security. In *SP '83: Proceedings of the 1983 IEEE Symposium on Security and Privacy*,

page 72, Washington, DC, USA, 1983. IEEE Computer Society.

[37] *Mandriva Linux*. `http://www.mandriva.com/en/community/users/club`.

[38] S. Micali. Fair public-key cryptosystems. In E. F. Brickell, editor, *Advances in Cryptology – CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 113–138, Aug. 16–20, 1992.

[39] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *International Conference on Very Large Data Bases*, pages 898–909, September 2003.

[40] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In J. Kilian, editor, *Advances in Cryptology – CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62, Santa Barbara, CA, USA, Aug. 19–23, 2001.

[41] D. Naor, A. Shenhav, and A. Wool. Toward securing untrusted storage without public-key operations. In *First International Workshop on Storage Security and Survivability*, november 2005.

[42] D. Reed and L. Svobodova. Swallow: A distributed data storage system for a local network. In A. West and P. Janson, editors, *Local Networks for Computer Communications*, pages 355–373. North-Holland Publ., Amsterdam, 1981.

[43] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[44] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.

[45] *Salon.com*. `http://www.salon.com/press/release/`.

[46] Q. Snell, A. Mikler, and J. Gustafson. Netpipe: A network protocol independent performace evaluator. In *IASTED International Conference on Intelligent Information Management and Systems*, 1996.

[47] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[48] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean. Self-healing key distribution with revocation. In *Proceedings of IEEE Symposium on Security and Privacy*, 2002.

[49] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.

[50] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. In *ISOC Network and Distributed System Security Symposium (NDSS 2004)*, 2004.

[51] A. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, 1982.