Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage

Kevin Fu
University of Massachusetts - Amherst

Seny Kamara
Johns Hopkins University

Tadayoshi Kohno
University of California - San Diego

Follow this and additional works at: https://scholarworks.umass.edu/cs_faculty_pubs

Recommended Citation
Fu, Kevin; Kamara, Seny; and Kohno, Tadayoshi, 'Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage' (2006). Computer Science Department Faculty Publication Series, 149. Retrieved from https://scholarworks.umass.edu/cs_faculty_pubs/149
Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage

Kevin Fu∗ Seny Kamara† Tadayoshi Kohno‡

November 30, 2005

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.

∗Department of Computer Science, 140 Governors Drive, University of Massachusetts, Amherst, MA 01003-9264, USA. E-Mail: kevinfu@cs.umass.edu. URL: http://www.cs.umass.edu/~kevinfu/. Supported in part by Project Oxygen and an Intel Fellowship. This research was performed while at The Johns Hopkins University and MIT.

†Department of Computer Science, The Johns Hopkins University, 3400 N. Charles Street, Baltimore, MD 21218, USA. E-Mail: seny@cs.jhu.edu. URL: http://www.cs.jhu.edu/~seny/. Supported by a Bell Labs Graduate Research Fellowship.

‡Department of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0404, USA. E-mail: tkohno@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/tkohno. Supported by an IBM Ph.D. Fellowship, NSF CCR-0208842, NSF ANR-0129617, and NSF CCR-0093337. Part of this research was performed while visiting the University of California at Berkeley.
## Contents

1 Introduction .............................................. 3
   1.1 Overview of contributions .......................... 3
   1.2 Related work ........................................ 5

2 Notation ................................................... 6

3 Problems with key rotation ............................. 6

4 Key Regression ............................................ 8
   4.1 Syntax and correctness requirements .................. 9
   4.2 Security goal ......................................... 10

5 Our preferred constructions ............................ 11
   5.1 The KR-SHA1 construction ........................... 11
   5.2 The KR-AES construction ............................. 12
   5.3 The KR-RSA construction ............................. 12
   5.4 Discussion ........................................... 13

6 Key regression from FSPRGs ............................ 13
   6.1 Forward-secure pseudorandom generators ............... 14
   6.2 An FSPRG-based key regression scheme ................. 14

7 Key regression from standard PRGs .................... 17
   7.1 FSPRGs from pseudorandom bit generators ............ 17
   7.2 A PRG-based key regression scheme .................. 18

8 The security of KR-AES .................................. 19

9 The security of KR-SHA1 ................................. 21

10 The security of KR-RSA ................................ 24
   10.1 Security of KR-RSA under one KeyderO oracle query 25
   10.2 Security under one KeyderO oracle query implies security under many 27
   10.3 The security of KR-RSA under multiple KeyderO oracle queries 30

11 Performance of key regression in access-controlled content distribution 30
   11.1 Hypothesis and methodology ........................ 31
   11.2 Microbenchmarks .................................... 32
   11.3 Secure content distribution on untrusted storage ....... 33

12 Conclusions .............................................. 35
1 Introduction

Content distribution networks (CDNs) such as Akamai [3], BitTorrent [15], and Coral [21] 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 [23, 26, 31, 33, 40, 43]. Under the lazy revocation model for access control [23, 33], 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 [33] 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 [33], 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 (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 $K_i$, 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 $K_i$, 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.\footnote{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.}

**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 $stm_l$ 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 [28] 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, 36].

**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

![Figure 1: Key regression overview; $stp_i$ and $stm_i$ respectively represent the $i$-th publisher and member states.](image-url)
decryption keys to members. On a simulated cable modem, a publisher using key regression can distribute 1,000 keys to 181 clients/sec 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 [38]. 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.

Versions. An extended abstract of this paper appears in the proceedings of the 2006 ISOC Network and Distributed System Security Symposium (NDSS) [25]. This is the full version, a copy of which is available at the IACR Cryptology ePrint Archive as report 2005/303 (http://eprint.iacr.org/). Part of this work also appears as Chapter 4 of [23].

1.2 Related work

The key rotation scheme in Plutus [33] inspired our research in key regression. Bellare and Yee [11] 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 \( \text{stm}_i \) 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

\[ \text{While Mandriva may wish to exercise access control over non-security-critical patches and upgrades, Mandriva would likely wish to allow all Mandriva users, including evicted Mandriva Club members, access to all security-critical patches. To enable such access, Mandriva could encrypt all security-critical patches with the key for the time period to which the patch is first applicable, or Mandriva could simply not encrypt security-critical patches.} \]
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. [14], one possible mechanism for distributing updated content encryption keys for a secure file system is to use a broadcast encryption scheme [18, 19, 20, 41]. 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 [51] 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 [33], 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 \( x \leftarrow Y \). If \( f \) is a deterministic (resp., randomized) function, then \( x \leftarrow f(y) \) (resp., \( x \leftarrow^* f(y) \)) 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 [44] key generator for some security parameter \( k \) is a randomized algorithm \( K_{\text{rsa}} \) that returns a triple \((N, e, d) \) rather than \((N, e, d) \leftarrow K_{\text{rsa}}(k) \). 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 \( ed \equiv 1 \mod \varphi(N) \), 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 [33] consists of three algorithms: setup, wndkey, and unwndkey. Figure 2 shows the original (RSA-based) Plutus key rotation scheme [33]. Following Plutus, and as Naor,
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 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 $\text{wndkey}(sk_i)$ 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 $\text{unwndkey}(K, pk)$ 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 $K_i$, 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 $K_l$ 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 $\mathcal{K}O$ and an encryption scheme $\mathcal{SE}$ 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 $\mathcal{K}O$, we start with a secure key rotation scheme $\mathcal{KO}$; $\mathcal{K}O$ outputs keys twice as long as $\mathcal{KO}$. The $\mathcal{K}O$ winding algorithm $\text{wndkey}$ invokes $\mathcal{KO}$’s winding algorithm to obtain a key $K$; $\text{wndkey}$ then returns $K||K$ as its key. On input a key $K||K$, $\text{unwndkey}$ invokes $\mathcal{KO}$’s unwinding algorithm with input $K$ to obtain a pseudorandom key.

\(^3\)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 $MW = 2$, $i = 1$, 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 $C$. 

---

### Algorithm setup

\[ \mathcal{K}_{\text{rsa}} \]

\[ (N, e, d) \leftarrow \mathcal{K}_{\text{rsa}} \quad \text{and} \quad K \leftarrow Z_N^* \]

\[ \text{pk} \leftarrow (N, e) \quad \text{and} \quad \text{sk} \leftarrow (K, N, d) \]

\[ \text{Return} \ (\text{pk}, \text{sk}) \]

### Algorithm wndkey

\[ \text{wndkey}(sk = (K, N, d)) \]

\[ K' \leftarrow K^d \mod N \]

\[ \text{sk}' \leftarrow (K', N, d) \]

\[ \text{Return} \ (K, \text{sk}') \]

### Algorithm unwndkey

\[ \text{unwndkey}(K, \text{pk} = (N, e)) \]

\[ \text{Return} \ K^e \mod N \]

---

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

---

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

---

For any user in possession of $K_i$, $i > 1$, and $pk$ can invoke $\text{unwndkey}(K, pk)$ 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 $K_i$, 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].
key $K'$; \texttt{unwndkey} then returns $K'||K'$ as its key. If the keys output by \texttt{wndkey} are unpredictable to evicted members, then so must be the keys output by \texttt{wndkey}. To build $\mathbb{SE}$, we start with a secure symmetric encryption scheme $\mathbb{SE}$; $\mathbb{SE}$ uses keys that are twice as long as $\mathbb{SE}$. The $\mathbb{SE}$ encryption and decryption algorithms take the key $K$, split it into two halves $K = L_1||L_2$, and run the respective algorithms of $\mathbb{SE}$ with keys $L_1\oplus L_2$. If the key $K$ is random, then the key $L_1\oplus L_2$ is random and $\mathbb{SE}$ runs the $\mathbb{SE}$ encryption algorithm with a uniformly selected random key. This means that $\mathbb{SE}$ satisfies the standard IND-CPA security goal if $\mathbb{SE}$ does.

Despite the individual security of both $\mathbb{KO}$ and $\mathbb{SE}$, when the keys output by $\mathbb{KO}$ are used to key $\mathbb{SE}$, the content publisher will always run $\mathbb{SE}$ 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 encrypted during time periods $l > i$ after being evicted. Although the construction of $\mathbb{KO}$ and $\mathbb{SE}$ 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, 28], 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 \textit{ad hoc} analyses since, under \textit{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 $\text{stp}_i$ 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 $\text{stm}_i$. As the arrows in Figure 1 suggest, given $\text{stm}_i$, 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 $\text{stm}_l$, $l > i$, 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 $\text{stm}_i$, 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 $\text{SHA1}$ as a random oracle and if the key rotation scheme produces unpredictable future keys $K_i$, then it might seem reasonable to conclude that an ex-member given $K_i$ should not be able to distinguish future values $\text{SHA1}(K_i)$, $l > i$, 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, $\text{KR-RSA}$ 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 SHA1($K_i$) 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 SHA1($K_i$) 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 $KR = (\text{setup, wind, unwind, keyder})$. 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 $\text{stp} \xrightarrow{\$} \text{setup}^H$, the randomized setup algorithm returns a publisher state. Via $(\text{stp}', \text{stm}) \xrightarrow{\$} \text{wind}^H(\text{stp})$, the randomized winding algorithm takes a publisher state $\text{stp}$ and returns a pair of publisher and member states or the error code ($\bot, \bot$). Via $\text{stm}' \xleftarrow{\$} \text{unwind}^H(\text{stm})$ the deterministic unwinding algorithm takes a member state $\text{stm}$ and returns a member state or the error code $\bot$. Via $K \xleftarrow{\$} \text{keyder}^H(\text{stm})$ the deterministic key derivation algorithm takes a member state $\text{stm}$ and returns a key $K \in \text{DK}$, where DK is the derived key space for $KR$. Let $MW \in \{1, 2, \ldots\} \cup \{\infty\}$ denote the maximum number of derived keys that $KR$ is designed to produce. We do not define the behavior of the algorithms when input the error code $\bot$.

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

Remarks. Although we allow $\text{wind}$ to be randomized, the $\text{wind}$ algorithms in all of our constructions are deterministic. We allow $\text{wind}$ to return ($\bot, \bot$) since we only require that $\text{wind}$ return non-error states for its first $MW$ invocations. We use the pair ($\bot, \bot$), rather than simply $\bot$, to denote an error from $\text{wind}$ since doing so makes our pseudocode cleaner. We allow $\text{unwind}$ to return $\bot$ since the behavior of $\text{unwind}$ 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 $\text{setup}$. We stress that $MW$ is a correctness parameter of $KR$, not a security parameter, meaning that even though the correctness criteria must hold for $MW$ invocations of $\text{wind}$, the security goal may not. One can also further generalize our definition and allow $\text{unwind}$ and $\text{keyder}$ 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 \( \text{stm}_l, l > i \), 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 \( \text{WindO} \) oracle). The \( \text{WindO} \) 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 \( \text{KeyderO} \). The key derivation oracle will either return real derived keys (via internal calls to \( \text{wind} \) and \( \text{keyder} \)) or random keys. The adversary’s goal is to guess whether the \( \text{KeyderO} \) oracle’s responses are real derived keys or random keys.

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

**Oracle WindO**
- \( i \leftarrow i + 1 \)
- If \( i > MW \) then return \( \perp \)
- \( (\text{stp}, \text{stm}) \leftarrow \text{wind}^H(\text{stp}) \)
- Return \( \text{stm} \)

**Oracle KeyderO_b**
- \( i \leftarrow i + 1 \)
- If \( i > MW \) then return \( \perp \)
- \( (\text{stp}, \text{stm}) \leftarrow \text{wind}^H(\text{stp}) \)
- If \( b = 1 \) then \( K \leftarrow \text{keyder}^H(\text{stm}) \)
- If \( b = 0 \) then \( K \leftarrow \text{DK} \)
- Return \( K \)

The \( \text{KR-advantage} \) of \( \mathcal{A} \) in breaking the security of \( \mathcal{KR} \) is defined as

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

Under the concrete security approach [8], we say that \( \mathcal{KR} \) is \( KR\text{-secure} \) 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 \( \text{WindO} \) and \( \text{KeyderO} \) 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 \( \text{WindO} \) and \( \text{KeyderO} \) and instead ask that the underlying \( \text{wind} \) 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 \( \text{WindO} \) and \( \text{KeyderO} \) 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 \( \text{WindO} \) 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, 11] 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) [11].

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 $\text{stm}$ variable always denotes a 160-bit string.

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

<table>
<thead>
<tr>
<th>Alg. setup</th>
<th>Alg. wind(stp)</th>
<th>Alg. unwind(stm)</th>
<th>Alg. keyder(stm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{stm}_{\text{MW}} \leftarrow {0, 1}^{160}$</td>
<td>If $\text{stp} = \bot$ then return $(\bot, \bot)$</td>
<td>$\text{stm}' \leftarrow \text{SHA1}(\text{stm})$</td>
<td>$K \leftarrow \text{SHA1}(0^8</td>
</tr>
<tr>
<td>For $i = \text{MW}$ downto 2 do</td>
<td>Parse $\text{stp}$ as $\langle i, \text{stm}<em>1, \ldots, \text{stm}</em>{\text{MW}} \rangle$</td>
<td>Return $\text{stm}'$</td>
<td>Return $K$</td>
</tr>
<tr>
<td>$\text{stm}_{i-1} \leftarrow \text{unwind}(\text{stm}_i)$</td>
<td>If $i &gt; \text{MW}$ return $(\bot, \bot)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{stp} \leftarrow \langle 1, \text{stm}<em>1, \ldots, \text{stm}</em>{\text{MW}} \rangle$</td>
<td>$\text{stp}' \leftarrow \langle i + 1, \text{stm}<em>1, \ldots, \text{stm}</em>{\text{MW}} \rangle$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return $\text{stp}$</td>
<td>Return $(\text{stp}', \text{stm}_i)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>MW = $\infty$</th>
<th>KR-SHA1</th>
<th>KR-AES</th>
<th>KR-RSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random oracles</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>setup cost</td>
<td>MW SHA1 ops</td>
<td>MW AES ops</td>
<td>1 RSA key generation</td>
</tr>
<tr>
<td>wind cost</td>
<td>no crypto</td>
<td>no crypto</td>
<td>1 RSA decryption</td>
</tr>
<tr>
<td>unwind cost</td>
<td>1 SHA1 op</td>
<td>1 AES op</td>
<td>1 RSA encryption</td>
</tr>
<tr>
<td>keyder cost</td>
<td>1 SHA1 op</td>
<td>1 AES op</td>
<td>1 SHA1 op</td>
</tr>
</tbody>
</table>

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.
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 SHA1(·) in unwind as one random oracle and the application of SHA1$(0^8 || ·)$ in keyder as another random oracle. The proof of security for KR-SHA1 is thus in the random oracle model [9].

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, 36].

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

```
Alg. setup
stm_{MW} \leftarrow \{0, 1\}^{128}
For i = MW downto 2 do
    stm_{i-1} \leftarrow unwind(stm_{i})
stp \leftarrow (1, stm_1, \ldots, stm_{MW})
Return stp
```

```
Alg. wind(stp)
If stp = ⊥ then return (⊥, ⊥)
Parse stp as ⟨i, stm_1, \ldots, stm_{MW}⟩
If i > MW return (⊥, ⊥)
stp' \leftarrow (i + 1, stm_1, \ldots, stm_{MW})
Return (stp', stm_{i})
```

```
Alg. unwind(stm)
stm' \leftarrow AES_{stm}(0^{128})
Return stm'
```

```
Alg. keyder(stm)
K \leftarrow AES_{stm}(1^{128})
Return K
```

The derived key space for KR-AES is 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) [11]; we call our construction KR-FSPRG. We then recall one of Bellare and Yee’s [11] methods (FSPRG-PRG) for building secure FSPRGs from standard pseudorandom bit generators (PRGs) [11, 12, 54]. 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 AES_{stm}(0^{128})||AES_{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 stm \in \{0, 1\}^{160}, outputs SHA1(stm)||SHA1(0^8||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 MW = ∞, 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 KR-RSA = (setup, wind, unwind, keyder) is defined as follows. Let \( \mathcal{K}_{rsa} \) be an RSA key generator for some security parameter \( k \) and let \( m: \mathbb{Z}_{2^k} \to \{0, 1\}^k \) denote the standard big-endian encoding of the integers in \( \mathbb{Z}_{2^k} \) to \( k \)-bit strings.

\[
\text{Alg. setup} \quad (N, e, d) \leftarrow \mathcal{K}_{rsa} \\
S \leftarrow \mathbb{Z}_N^* \\
\text{stp} \leftarrow \langle N, e, d, S \rangle \\
\text{Return stp}
\]

\[
\text{Alg. wind(stp)} \\
\text{Parse stp as } \langle N, e, d, S \rangle \\
S' \leftarrow S^d \mod N \\
\text{stp}' \leftarrow \langle N, e, d, S' \rangle \\
\text{stm} \leftarrow \langle N, e, S \rangle \\
\text{Return } (\text{stp}', \text{stm})
\]

\[
\text{Alg. unwind(stm)} \\
\text{Parse stm as } \langle N, e, S \rangle \\
S' \leftarrow S^e \mod N \\
\text{stm}' \leftarrow \langle N, e, S' \rangle \\
\text{Return stm}'
\]

\[
\text{Alg. keyder(stm)} \\
\text{Parse stm as } \langle N, e, S \rangle \\
K \leftarrow \text{SHA1}(m(S)) \\
\text{Return } K
\]

The derived key space for KR-RSA is \( \text{DK} = \{0, 1\}^{160} \). In our experiments, we set \( k = 1024 \), and \( \mathcal{K}_{rsa} \) 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, 37, 39], 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 [52] 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 [32, 34] 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 [16].

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) [11]. 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 [11] 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: \( SBG = (\text{seed, next}) \). The randomized setup algorithm returns an initial state; we write this as \( \text{stg} \leftarrow \text{seed} \). The deterministic next step algorithm takes a state as input and returns a new state and an output from \( \text{OutSp}_{SBG} \); or the pair \((\bot, \bot)\); we write this as \( (\text{stg}', K) \leftarrow \text{next}(\text{stg}) \). We require that the set \( \text{OutSp}_{SBG} \) is efficiently samplable. \( \text{MaxLen}_{SBG} \in \{1, 2, \ldots \} \cup \{\infty\} \) denotes the maximum number of output blocks that \( SBG \) is designed to produce from a correctness (not security) perspective.

Correctness. The correctness requirement for stateful PRGs is as follows: let \( \text{stg}_0 \leftarrow \text{seed} \) and, for \( i = 1, 2, \ldots \), let \( (\text{stg}_i, K_i) \leftarrow \text{next}(\text{stg}_{i-1}) \). We require that for \( i \leq \text{MaxLen}_{SBG} \), \( (\text{stg}_i, K_i) \neq (\bot, \bot) \).

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

<table>
<thead>
<tr>
<th>Experiment ( \text{Exp}_{SBG,A}^{\text{fsprg}=b} )</th>
<th>Oracle ( \text{NextO}_b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stg} \leftarrow \text{seed} )</td>
<td>( (\text{stg}, K) \leftarrow \text{next}(\text{stg}) )</td>
</tr>
<tr>
<td>( \text{st} \leftarrow A^{\text{NextO}_b}(\text{find}) )</td>
<td>If ( b = 0 ) then ( K \leftarrow \text{OutSp}_{SBG} )</td>
</tr>
<tr>
<td>( \text{b}' \leftarrow A(\text{guess, stg, st}) )</td>
<td>Return ( K )</td>
</tr>
<tr>
<td>Return ( \text{b}' )</td>
<td></td>
</tr>
</tbody>
</table>

The \( \text{FSPRG-advantage} \) of \( A \) in breaking the security of \( SBG \) is defined as

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

Under the concrete security approach, the scheme \( SBG \) is said to be \( \text{FSPRG-secure} \) if the FSPRG-advantage of all adversaries \( A \) 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 \( SBG \) 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 (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 \( \text{stp} \) consisting of an index \( i \), initially 1, and the MW member states. The wind algorithm, on input a publisher state \( \text{stp} \) with index \( i \leq MW \), outputs the \( i \)-th member state in the vector and outputs a revised publisher state \( \text{stp}' \) with index \( i + 1 \). On input a member state \( \text{stm} \), the wind and keyder algorithms both invoke next on \( \text{stm} \) to obtain a pair \( (\text{stm}', K) \); wind then outputs the revised member state \( \text{stm}' \) whereas keyder outputs the key \( K \).
Construction 6.1 [KR-FSPRG] Given a stateful generator $SBG = (seed, next)$, we can construct a key regression scheme $KR$-FSPRG = (setup, wind, unwind, keyder) as follows. $MW \leq \text{MaxLen}_{SBG}$ is a positive integer and a parameter of the construction.

\begin{align*}
\text{Alg. setup} & \quad \text{Alg. wind(stp)} & \quad \text{Alg. unwind(stm)} \\
\text{stg}_{MW} \leftarrow \text{seed} & \quad \text{If stp} = \bot \quad \text{Parse stp as } \langle \text{stm}, K, \text{stp} \rangle & \quad \text{Return } (\text{stm}', K) \\
\text{For } i = MW \text{ downto } 2 \text{ do } & \quad \langle i, \text{stg}_1, \dotsc, \text{stg}_{MW} \rangle & \quad \text{Return } (\text{stm}', \text{stg}_i) \\
\text{stp} \leftarrow \langle 1, \text{stg}_1, \dotsc, \text{stg}_{MW} \rangle & \quad \text{If } i > MW \text{ return } (\bot, \bot) & \quad \text{Adv}_{SBG,B}^{\text{fsprg}}
\end{align*}

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

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 $SBG$ is a secure forward-secure pseudorandom bit generator (i.e., is FSPRG-secure), then the resulting key regression scheme $KR$-FSPRG built from $SBG$ via Construction 6.1 will be secure (i.e., KR-secure). Specifically, Theorem 6.2 says that given an adversary $A$ against KR-FSPRG, one can construct an adversary $B$ against $SBG$ such that $B$ uses reasonable resources (if $A$ 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 $A$ makes. These properties imply security for KR-FSPRG since, if $SBG$ is FSPRG-secure and if $A$ uses reasonable resources, then $\text{Adv}_{SBG,B}^{\text{fsprg}}$ and $q$ must both be small, which means that $\text{Adv}_{KR,A}^{\text{kr}}$, the advantage of $A$ in attacking KR-FSPRG, must be small as well.

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

\[
\text{Adv}_{KR,A}^{\text{kr}} \leq (q + 1) \cdot \text{Adv}_{SBG,B}^{\text{fsprg}}
\]

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

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

Proof of Theorem 6.2: The adversary $B$ is shown in Figure 4. The main idea is that if $B$ correctly guesses the number of WindO queries that $A$ will make, then $B$’s simulation is perfect for either choice of bit $b$. If $B$ does not correctly guess the number of WindO oracle queries, then it always returns 0, regardless of the value of the bit $b$. We restrict $q$ to the minimum of $MW$ and the maximum number of wind and key derivation oracle queries that $A$ makes since wind is defined to return $(\bot, \bot)$ after $MW$ invocations.
Formally, we claim that
\[
\Pr \left[ \Exp_{KR,A}^{kr-1} = 1 \right] = (q + 1) \cdot \Pr \left[ \Exp_{SBG,B}^{fsprg-1} = 1 \right] \quad (2)
\]
\[
\Pr \left[ \Exp_{KR,A}^{kr-0} = 1 \right] = (q + 1) \cdot \Pr \left[ \Exp_{SBG,A}^{fsprg-0} = 1 \right], \quad (3)
\]
from which it follows that
\[
\Adv_{KR,A}^{kr} = \Pr \left[ \Exp_{KR,A}^{kr-1} = 1 \right] - \Pr \left[ \Exp_{KR,A}^{kr-0} = 1 \right]
= (q + 1) \cdot \left( \Pr \left[ \Exp_{SBG,B}^{fsprg-1} = 1 \right] - \Pr \left[ \Exp_{SBG,A}^{fsprg-0} = 1 \right] \right)
\leq (q + 1) \cdot \Adv_{SBG,B}^{fsprg}
\]
as desired.

It remains to justify Equation (2), Equation (3), and the resources of \( \mathcal{B} \). Let \( \mathcal{E}_1 \) and \( \mathcal{E}_0 \) respectively denote the events that \( \mathcal{B} \) sets \( \text{bad} \) to \( \text{true} \) in the experiments \( \Exp_{SBG,B}^{fsprg-1} \) and \( \Exp_{SBG,A}^{fsprg-0} \), i.e., when \( \mathcal{B} \) fails to correctly guess the number of wind oracle queries that \( \mathcal{A} \) makes. Let \( \Pr_1 [ \cdot ] \) and \( \Pr_0 [ \cdot ] \)
respectively denote probabilities over $\text{Exp}^{\text{fsprg}^{-1}}_{\text{SBG}, B}$ and $\text{Exp}^{\text{fsprg}^{-0}}_{\text{SBG}, A}$. We now claim that

$$\Pr \left[ \text{Exp}^{\text{kr}-1}_{\text{KR}, A} = 1 \right] = \Pr \left[ \text{Exp}^{\text{fsprg}^{-1}}_{\text{SBG}, B} = 1 \mid \mathcal{E}_1 \right]$$

(4)

$$= \Pr \left[ \text{Exp}^{\text{fsprg}^{-1}}_{\text{SBG}, B} = 1 \land \mathcal{E}_1 \right] \cdot \frac{1}{\Pr_1 \left[ \mathcal{E}_1 \right]}$$

(5)

$$= (q + 1) \cdot \Pr \left[ \text{Exp}^{\text{fsprg}^{-1}}_{\text{SBG}, B} = 1 \land \mathcal{E}_1 \right]$$

(6)

$$= (q + 1) \cdot \left( \Pr \left[ \text{Exp}^{\text{fsprg}^{-1}}_{\text{SBG}, B} = 1 \land \mathcal{E}_1 \right] + \Pr \left[ \text{Exp}^{\text{fsprg}^{-1}}_{\text{SBG}, B} = 1 \land \mathcal{E}_1 \right] \right)$$

(7)

Equation (4) is true because when the event $\mathcal{E}_1$ does not occur, i.e., when $\mathcal{B}$ correctly guesses the number of wind oracle queries that $\mathcal{A}$ will make, then $\mathcal{B}$ in $\text{Exp}^{\text{fsprg}^{-1}}_{\text{SBG}, B}$ runs $\mathcal{A}$ exactly as $\mathcal{A}$ would be run in $\text{Exp}^{\text{kr}-1}_{\text{KR}, A}$. Equation (5) follows from conditioning off $\Pr_1 \left[ \mathcal{E}_1 \right]$ and Equation (6) is true because $\mathcal{B}$ chooses $q'$ from $q + 1$ possible values and therefore $\Pr_1 \left[ \mathcal{E}_1 \right] = 1/(q + 1)$. To justify Equation (7), note that $\Pr \left[ \text{Exp}^{\text{fsprg}^{-1}}_{\text{SBG}, B} = 1 \land \mathcal{E}_1 \right] = 0$ since $\mathcal{B}$ always returns 0 whenever it fails to correctly guess the number of wind oracle queries that $\mathcal{A}$ will make. This justifies Equation (2).

To justify Equation (3), note that

$$\Pr \left[ \text{Exp}^{\text{kr}-0}_{\text{KR}, A} = 1 \right] = \Pr \left[ \text{Exp}^{\text{fsprg}^{-0}}_{\text{SBG}, B} = 1 \mid \mathcal{E}_0 \right]$$

since when the event $\mathcal{E}_0$ does not occur, $\mathcal{B}$ in $\text{Exp}^{\text{fsprg}^{-0}}_{\text{SBG}, B}$ runs $\mathcal{A}$ exactly as $\mathcal{A}$ would be run in $\text{Exp}^{\text{kr}-0}_{\text{KR}, A}$. The remaining justification for Equation (3) is analogous to our justification of Equation (2) above.

The resources for $\mathcal{B}$ is within a small constant factor of the resources for $\mathcal{A}$ except that $\mathcal{B}$ must execute the setup algorithm itself, which involves querying its oracle up to MW times.

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 [11] 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) [11, 12, 54] is a function $G : \{0, 1\}^k \rightarrow \{0, 1\}^{k+l}$ 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 $A$ is an adversary, we let

$$\text{Adv}_{A} \left[ K \xleftarrow{\$} \{0, 1\}^k ; x \leftarrow G(K) : A(x) = 1 \right] - \text{Adv}_{A} \left[ x \xleftarrow{\$} \{0, 1\}^{k+l} : A(x) = 1 \right]$$

denote the PRG-advantage of $A$ in attacking $G$. Under the concrete security approach, $G$ is said to be a “secure PRG” if the PRG-advantage of all adversaries $A$ using reasonable resources is “small.”
A PRG-based FSPRG. Bellare and Yee [11] 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 $\text{stg}$, computes the $(k + l)$-bit string $G(\text{stg})$ and outputs the first $k$ bits of $G(\text{stg})$ as the next state and the remaining $l$ bits as the key.

**Construction 7.1 [FSPRG-PRG, Construction 2.2 of [11].]** Given a PRG $G : \{0, 1\}^k \rightarrow \{0, 1\}^{k+l}$ we can construct a FSPRG $\text{SBG} = (\text{seed}, \text{next})$ as shown below

<table>
<thead>
<tr>
<th>Alg.</th>
<th>seed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{stg}_0 \overset{$}{\leftarrow} {0, 1}^k$</td>
<td>Alg. $\text{next}(\text{stg}_i)$</td>
</tr>
<tr>
<td>return $\text{stg}_0$</td>
<td>$r \overset{$}{\leftarrow} G(\text{stg}_i)$</td>
</tr>
<tr>
<td></td>
<td>$\text{stg}_{i+1} \leftarrow \text{first } k \text{ bits of } r$</td>
</tr>
<tr>
<td></td>
<td>$K \leftarrow \text{last } l \text{ bits of } r$</td>
</tr>
<tr>
<td></td>
<td>return $(\text{stg}_{i+1}, K)$</td>
</tr>
</tbody>
</table>

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

The following lemma comes from Bellare and Yee [11] 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 $\text{Adv}^{\text{prg}}_{G,B}$ must be small for all adversaries $B$ using reasonable resources. Further, if an adversary $A$ against FSPRG-PRG uses reasonable resources, then the number of oracle queries $q$ that it makes must also be small and $B$ must also use reasonable resources. These properties, coupled with Equation (8), means that the advantage of all adversaries $A$ 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 [11].]** Let $G : \{0, 1\}^k \rightarrow \{0, 1\}^{k+l}$ be a PRG, and let $\text{SBG}$ be the FSPRG built using $G$ according to Construction 7.1. Given an adversary $A$ attacking $\text{SBG}$ that makes at most $q$ queries to its oracle, we can construct an adversary $B$ such that

$$\text{Adv}^{\text{prg}}_{\text{SBG}, A} \leq 2q \cdot \text{Adv}^{\text{prg}}_{G,B}$$

where $B$ uses within a small constant factor of the resources of adversary $A$ 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 $G : \{0, 1\}^k \rightarrow \{0, 1\}^{k+l}$ be a pseudorandom bit generator. We can construct a key regression scheme KR-PRG = (setup, wind, unwind, keyder) from $G$ as follows. $\text{MW}$ is a positive integer and a parameter of the construction.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{stm}_{\text{MW}} \overset{$}{\leftarrow} {0, 1}^k$</td>
<td>Alg. $\text{wind}(\text{stm})$</td>
</tr>
<tr>
<td>For $i = \text{MW}$ downto 2 do</td>
<td>If $\text{stm} = \bot$ then return $(\bot, \bot)$</td>
</tr>
<tr>
<td>$\text{stm}_{i-1} \leftarrow \text{unwind}(\text{stm}_i)$</td>
<td>Parse $\text{stm}$ as</td>
</tr>
<tr>
<td>$\text{stm} \leftarrow \langle 1, \text{stm}<em>1, \ldots, \text{stm}</em>{\text{MW}} \rangle$</td>
<td>$(i, \text{stm}<em>1, \ldots, \text{stm}</em>{\text{MW}})$</td>
</tr>
<tr>
<td>Return $\text{stm}$</td>
<td>If $i &gt; \text{MW}$ return $(\bot, \bot)$</td>
</tr>
<tr>
<td></td>
<td>$\text{stm}' \leftarrow \langle i+1, \text{stm}<em>1, \ldots, \text{stm}</em>{\text{MW}} \rangle$</td>
</tr>
<tr>
<td></td>
<td>Return $(\text{stm}', \text{stm}_i)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg.</th>
<th>unwind($\text{stm}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leftarrow G(\text{stm})$</td>
<td>Alg. keyder($\text{stm}$)</td>
</tr>
<tr>
<td>$\text{stm}' \leftarrow \text{first } k \text{ bits of } x$</td>
<td>$x \leftarrow G(\text{stm})$</td>
</tr>
<tr>
<td>Return $\text{stm}'$</td>
<td>$K \leftarrow \text{last } l \text{ bits of } x$</td>
</tr>
<tr>
<td></td>
<td>Return $K$</td>
</tr>
</tbody>
</table>
The derived key space for KR-PRG is $DK = \{0,1\}^l$.

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 $A$ attacking KR-PRG that uses reasonable resources, and assuming that MW is small, one can construct an adversary $B$ against $G$ such that $B$ uses reasonable resources and Equation (9) in the theorem statement holds; $q$ is the minimum of MW and the maximum number of wind and key derivation oracle queries that $A$ makes. These properties imply security for KR-PRG since, if $G$ is PRG-secure and since $A$ uses reasonable resources, $Adv_{G,A}^{prg}$ and $q$ must both be small, which means that $Adv_{KR,A}^{kr}$, the advantage of $A$ in attacking KR-PRG, must be small as well.

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

$$Adv_{KR,A}^{kr} \leq 2 \cdot (q + 1)^2 \cdot Adv_{G,B}^{prg}$$

where $q$ is the minimum of MW and the maximum number of queries $A$ makes to its WindO and KeyderO oracles. Adversary $B$ uses within a small constant factor of the resources of $A$, 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 $G(X) = AES_X(0^{128})||AES_X(1^{128})$ 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, 36].

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

$$Adv_{E,A}^{prp} = \Pr [K \xleftarrow{\$} \{0,1\}^k : A^{E_K()} = 1] - \Pr [g \xleftarrow{\$} \text{Perm}(l) : A^{g()} = 1]$$

denote the **PRP-advantage** of $A$ in attacking $E$. Under the concrete security approach, $E$ is said to be a “secure PRP” if the PRP-advantage of all adversaries $A$ 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

\[
\text{Adv}_{\text{KR,}\mathcal{A}}^{\text{kr}} \leq 2 \cdot (q + 1)^2 \cdot \left( \text{Adv}_{\text{AES,}\mathcal{B}}^{\text{prp}} + 2^{-128} \right)
\]

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 \( \text{Adv}_{\text{AES,}\mathcal{B}}^{\text{prp}} \) must be small. If both \( q \) and \( \text{Adv}_{\text{AES,}\mathcal{B}}^{\text{prp}} \) are small, then by Equation (10) \( \text{Adv}_{\text{KR,}\mathcal{A}}^{\text{kr}} \) 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 (10) becomes

\[
\text{Adv}_{\text{KR,}\mathcal{A}}^{\text{kr}} \leq 2^{42} \cdot \text{Adv}_{\text{AES,}\mathcal{B}}^{\text{prp}} + 2^{-86},
\]

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 \( G(X) = AES_X(0^{128})||AES_X(1^{128}) \), \( X \in \{0,1\}^{128} \), is a secure PRG if AES is a secure PRP. Details follow.

**Proof of Theorem 8.1:** To instantiate KR-AES from Construction 7.3, we set \( k = l = 128 \) and, for any \( X \in \{0,1\}^{128} \), we define \( G \) as \( G(X) = AES_X(0^{128})||AES_X(1^{128}) \).

We first claim that, given an adversary \( \mathcal{B} \) attacking \( G \), we can construct an adversary \( \mathcal{C} \) attacking AES such that

\[
\text{Adv}_{G,\mathcal{B}}^{\text{prp}} \leq \text{Adv}_{AES,\mathcal{C}}^{\text{prp}} + 2^{-128}
\]

and \( \mathcal{C} \) makes two oracle queries and uses within a small constant factor of the resources of \( \mathcal{B} \). Theorem 8.1 follows from this claim and Theorem 7.4.

We now justify our claim above. Let Func\((l,l)\) denote the set of all functions from \( \{0,1\}^l \) to \( \{0,1\}^l \). Let \( \mathcal{C} \) be a PRP adversary that runs \( \mathcal{B} \) with input \( f(0^{128})||f(1^{128}) \), where \( f: \{0,1\}^{128} \rightarrow \{0,1\}^{128} \) is \( \mathcal{C} \)'s oracle. Adversary \( \mathcal{C} \) then returns the same bit that \( \mathcal{B} \) returns.

Note that

\[
\Pr \left[ K \overset{\$}{\leftarrow} \{0,1\}^{128} ; x \leftarrow G(K) : \mathcal{B}(x) = 1 \right] = \Pr \left[ K \overset{\$}{\leftarrow} \{0,1\}^{128} ; C^{AES_K(\cdot)} = 1 \right]
\]

since, when \( \mathcal{C} \)'s oracle is AES\(_K(\cdot)\), \( \mathcal{C} \) runs \( \mathcal{B} \) with input AES\(_K(0^{128})||AES_K(1^{128}) \), for a randomly selected key \( K \), which has the same distribution as \( G(K) \) for a randomly selected key \( K \). Addi-
tionally,
\[
\Pr \left[ x \leftarrow \{0,1\}^{256} : B(x) = 1 \right] = \Pr \left[ g \leftarrow \text{Func}(128, 128) : C^g(\cdot) = 1 \right]
\]
since, when \(C\)'s oracle is a random function from \(\{0,1\}^{128}\) to \(\{0,1\}^{128}\), it runs \(B\) with a random 256-bit string.

Expanding the definition of \(\text{Adv}^{\text{PRG}}_{G,B}\) and substituting the above equalities, we have
\[
\text{Adv}^{\text{PRG}}_{G,B} = \Pr \left[ K \leftarrow \{0,1\}^{128} : x \leftarrow G(K) : B(x) = 1 \right] - \Pr \left[ x \leftarrow \{0,1\}^{256} : B(x) = 1 \right]
\]
\[= \Pr \left[ K \leftarrow \{0,1\}^{128} : C^{\text{AES}_K(\cdot)} = 1 \right] - \Pr \left[ g \leftarrow \text{Func}(128, 128) : C^{g(\cdot)} = 1 \right].\]

If we subtract and add \(\Pr \left[ g \leftarrow \text{Perm}(128) : C^{g(\cdot)} = 1 \right]\) and apply the definition of \(\text{Adv}^{\text{PRP}}_{\text{AES},C}\), we get
\[
\text{Adv}^{\text{PRG}}_{G,B} = \Pr \left[ K \leftarrow \{0,1\}^{128} : C^{\text{AES}_K(\cdot)} = 1 \right] - \Pr \left[ g \leftarrow \text{Perm}(128) : C^{g(\cdot)} = 1 \right] - \Pr \left[ g \leftarrow \text{Func}(128, 128) : C^{g(\cdot)} = 1 \right]
\]
\[+ \Pr \left[ g \leftarrow \text{Perm}(128) : C^{g(\cdot)} = 1 \right] - \Pr \left[ g \leftarrow \text{Func}(128, 128) : C^{g(\cdot)} = 1 \right].\]

Using the standard PRF/PRP switching result from [8], re-proven in [10, 48], and the fact that \(C\) makes only two oracle queries, the above simplifies to Equation (11), completing the proof.

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 \(G(X) = \text{SHA1}(0^8 || X)\) 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 SHA1(\(\cdot\)) in KR-SHA1’s unwind algorithm and 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 SHA1(\(\cdot\)) and 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 \(H_1: \{0,1\}^k \rightarrow \{0,1\}^k\) and \(H_2: \{0,1\}^k \rightarrow \{0,1\}^l\) be random oracles. We can construct a key regression scheme \(\text{KR-RO} = (\text{setup}, \text{wind}, \text{unwind}, \text{keyder})\) from \(H_1\) and \(H_2\) as shown below. MW is a positive integer and a parameter of the construction.
Proof of Theorem 9.2:
Consider the experiments to

\[ KR \]

Formally, let \( H \)

that the advantage of \( A \)

As a concrete example of the bound in Theorem 9.2, consider the case where

that there exist two distinct indices \( i, j \)

where \( \text{Exp} \) is the maximum number of queries total that adversary \( A \)

The derived key space for \( KR \) is secure in the random oracle model.

We claim that

\[ \text{Adv}_{KR,A}^{kr} \leq \frac{(MW)^2}{2^{k+1}} + \frac{q \cdot MW}{2^k - MW - q}, \tag{12} \]

we have that

\[ \text{Adv}_{KR,A}^{kr} \leq \frac{(MW)^2}{2^{k+1}} + \frac{q \cdot MW}{2^k - MW - q}, \]

where \( q \) is the maximum number of queries total that adversary \( A \)

makes at most \( q = 2^{40} \) queries to its random oracles. Then Equation (12) tells us

that the advantage of \( A \) 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 below, 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.

Proof of Theorem 9.2: Consider the experiments \( \text{Exp}_{KR,A}^{kr-1} \) and \( \text{Exp}_{KR,A}^{kr-0} \). Let \( stm_1, stm_2, \ldots, stm_{MW} \) denote the member states as computed by \( \text{setup} \), and let \( w' \) denote the variable number

of WindO oracle queries that \( A \) made in its \text{member} stage. Let \( \mathcal{E}_1 \) be the event in \( \text{Exp}_{KR,A}^{kr-1} \)

that \( w' \leq MW - 1 \) and that \( A \) queries either its \( H_1 \) or \( H_2 \) random oracles with some string

\( x \in \{stm_{w'+1}, \ldots, stm_{MW}\} \). Let \( \mathcal{E}_0 \) be the event in \( \text{Exp}_{KR,A}^{kr-0} \) that \( w' \leq MW - 1 \) and that \( A \) queries either its \( H_1 \) or \( H_2 \) random oracles with some string \( x \in \{stm_{w'+1}, \ldots, stm_{MW}\} \). Let \( \mathcal{F}_1 \) be the event in \( \text{Exp}_{KR,A}^{kr-1} \) that there exist two distinct indices \( i, j \in \{1, \ldots, MW\} \) such that \( stm_i = stm_j \) and let \( \mathcal{F}_0 \) be the event in \( \text{Exp}_{KR,A}^{kr-0} \) that there exist two distinct indices \( i, j \in \{1, \ldots, MW\} \) such that \( stm_i = stm_j \).

We claim that

\[ \text{Adv}_{KR,A}^{kr} \leq \Pr \left[ \text{Exp}_{KR,A}^{kr-1} = 1 \land \mathcal{F}_1 \right] + \Pr \left[ \text{Exp}_{KR,A}^{kr-1} = 1 \land \mathcal{E}_1 \land \overline{\mathcal{F}_1} \right], \tag{13} \]

that

\[ \Pr \left[ \text{Exp}_{KR,A}^{kr-1} = 1 \land \mathcal{F}_1 \right] \leq \frac{(MW)^2}{2^{k+1}}, \tag{14} \]
If we consider the event $F$, and that $Exp_{K,R,A}^{kr-1}$ and $Exp_{K,R,A}^{kr-0}$ respectively. From Definition 4.1, we have

$$Adv_{K,R,A}^{kr} = Pr\left[Exp_{K,R,A}^{kr-1} = 1\right] - Pr\left[Exp_{K,R,A}^{kr-0} = 1\right]$$

$$= Pr\left[Exp_{K,R,A}^{kr-1} = 1 \land F_1\right] + Pr\left[Exp_{K,R,A}^{kr-1} = 1 \land \overline{E}_1 \land \overline{F}_1\right]$$

$$+ Pr\left[Exp_{K,R,A}^{kr-0} = 1 \land \overline{E}_0 \land \overline{F}_0\right] - Pr\left[Exp_{K,R,A}^{kr-0} = 1 \land \overline{E}_0 \land \overline{F}_0\right]$$

$$\leq Pr\left[Exp_{K,R,A}^{kr-1} = 1 \land F_1\right] + Pr\left[Exp_{K,R,A}^{kr-1} = 1 \land \overline{E}_1 \land \overline{F}_1\right]$$

$$+ Pr\left[Exp_{K,R,A}^{kr-1} = 1 \land \overline{E}_1 \land \overline{F}_1\right] - Pr\left[Exp_{K,R,A}^{kr-0} = 1 \land \overline{E}_0 \land \overline{F}_0\right]. \quad (16)$$

By conditioning,

$$Pr\left[Exp_{K,R,A}^{kr-1} = 1 \land \overline{E}_1 \land \overline{F}_1\right] = Pr\left[Exp_{K,R,A}^{kr-1} = 1 \mid \overline{E}_1 \land \overline{F}_1\right] \cdot Pr_1 \left[\overline{E}_1 \land \overline{F}_1\right]$$

and

$$Pr\left[Exp_{K,R,A}^{kr-0} = 1 \land \overline{E}_0 \land \overline{F}_0\right] = Pr\left[Exp_{K,R,A}^{kr-0} = 1 \mid \overline{E}_0 \land \overline{F}_0\right] \cdot Pr_0 \left[\overline{E}_0 \land \overline{F}_0\right].$$

Prior to the adversary causing the events $E_1 \lor F_1$ and $E_0 \lor F_0$ to occur in their respective experiments, $A$'s view is identical in both experiments, meaning that

$$Pr_1 \left[\overline{E}_1 \land \overline{F}_1\right] = Pr_0 \left[\overline{E}_0 \land \overline{F}_0\right].$$

Similarly, if the events do not occur, then the outcome of $Exp_{K,R,A}^{kr-1}$ and $Exp_{K,R,A}^{kr-0}$ will be the same since the output of a random oracle is random if the input is unknown; i.e., the response to $A$'s key derivation oracle query in the non-member stage will be random in both cases and therefore

$$Pr\left[Exp_{K,R,A}^{kr-1} = 1 \mid \overline{E}_1 \land \overline{F}_1\right] = Pr\left[Exp_{K,R,A}^{kr-0} = 1 \mid \overline{E}_0 \land \overline{F}_0\right].$$

Consequently,

$$Pr\left[Exp_{K,R,A}^{kr-1} = 1 \land \overline{E}_1 \land \overline{F}_1\right] = Pr\left[Exp_{K,R,A}^{kr-0} = 1 \land \overline{E}_0 \land \overline{F}_0\right].$$

Combining the above equation with Equation (16) gives Equation (13).

Returning to Equation (14), we first note that

$$Pr\left[Exp_{K,R,A}^{kr-1} = 1 \land F_1\right] \leq Pr_1 \left[F_1\right].$$

If we consider the event $F_1$, we note that the setup algorithm selects the points $stm_{MW}$, $stm_{MW-1}$, $stm_{MW-2}$, and so on, uniformly at random from $\{0,1\}^k$ until a collision occurs. Since this is exactly the standard birthday paradox [8], we can upper bound $Pr_1 \left[F_1\right]$ as

$$Pr_1 \left[F_1\right] \leq \frac{(MW)^2}{2^{k+1}}.$$
The derived key space for KR cipher as a hash function in this manner in [17]. We choose to prove the security of KR defined as AES. We remark that in addition to viewing KR, to justify Equation (15), we begin by noting that Equation (14) follows.

Consider the adversary $A$ in $\text{Exp}_{KR,A}^{\perp}$ and assume that $\mathcal{F}_1$ does not occur. Consider any snapshot of the entire state of $\text{Exp}_{KR,A}^{\perp}$ before $A$ causes $\mathcal{E}_1$ to occur, and let $q'$ denote the number of $H_1$ and $H_2$ oracle queries that $A$ has made prior to the snapshot being taken. Then the member states $\text{stm}_{w'+1}, \ldots, \text{stm}_{MW}$ are restricted only in that they are distinct strings from $\{0,1\}^k$ and that none of the strings are from $\{\text{stm}_1, \ldots, \text{stm}_{w'}\}$ or the set of $A$’s $q'$ queries to its random oracles; i.e., the member states that $A$ obtained in its member stage and the responses from the KeyderO oracle do not reveal additional information to the adversary. This means that if the adversary’s next oracle query after this snapshot is to one of its random oracles, and if that input for that oracle query is some string $x$, then the probability that $x \in \{\text{stm}_{w'+1}, \ldots, \text{stm}_{MW}\}$, i.e., the probability that $A$’s oracle query would cause $\mathcal{E}_1$ to occur, is at most $(MW - w')/(2k - (w' + q')) \leq MW/(2k - MW - q')$. Summing over all of $A$’s $q$ random oracle queries and taking an upper bound, we have

$$\Pr[\mathcal{E}_1 | \mathcal{F}_1] \leq \frac{q \cdot MW}{2k - MW - q},$$

which completes the proof.

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 AES$_X(0^{128})$ and $H_2(X)$ defined as AES$_X(1^{128})$; Diffie and Hellman suggest using a block cipher as a hash function in this manner in [17]. 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 $K_{rsa}$ for some security parameter $k$ and a random oracle $H: \mathbb{Z}_{2^k} \to \{0,1\}^l$, we can construct a key regression scheme $\text{KR-RSA-RO} = (\text{setup, wind, unwind, keyder})$ as shown below, where $MW = \infty$.

<table>
<thead>
<tr>
<th>Alg. setup$^H$</th>
<th>Alg. wind$^H$(stp)</th>
<th>Alg. unwind$^H$(stm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(N, e, d) \xleftarrow{$} K_{rsa}$</td>
<td>Parse stp as $\langle N, e, d, S \rangle$</td>
<td>Parse stp as $\langle N, e, S \rangle$</td>
</tr>
<tr>
<td>$S \xleftarrow{$} \mathbb{Z}_N^*$</td>
<td>$S' \leftarrow S^d \mod N$</td>
<td>$S' \leftarrow S \mod N$</td>
</tr>
<tr>
<td>stp $\leftarrow \langle N, e, d, S \rangle$</td>
<td>stp $\leftarrow \langle N, e, d, S' \rangle$</td>
<td>stp $\leftarrow \langle N, e, S \rangle$</td>
</tr>
<tr>
<td>Return stp</td>
<td>stm $\leftarrow \langle N, e, S \rangle$</td>
<td>Return stp</td>
</tr>
</tbody>
</table>

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


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 KRsa 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 KRsa be an RSA key generator with security parameter k. If A is an adversary, we let

\[ \text{Adv}^{rsa\text{-ow}}_{\text{KRsa}, A} = \Pr \left[ (N, e, d) \xleftarrow{} \text{KRsa} ; x, y \leftarrow \mathbb{Z}_N^* \mid y = x^e \mod N : A(y, e, N) = 1 \right] \]

denote the RSA one-way advantage of A in inverting RSA with the key generator KRsa. Under the concrete security approach, KRsa is said to be a “one-way” if the RSA one-way advantage of all adversaries 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 KRsa 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 KRsa is an RSA key generator with security parameter k, then the key regression scheme KR built from KRsa 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 KRsa is one-way. Concretely, given an adversary A attacking KR that makes at most one key derivation oracle query, we can construct an adversary B attacking KRsa such that

\[ \text{Adv}^{kr}_{\text{KR}, A} \leq (q + 1) \cdot \text{Adv}^{rsa\text{-ow}}_{\text{KRsa}, B}, \quad (17) \]

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

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

**Proof of Lemma 10.2:** Consider the experiments \(\text{Exp}^{kr \leftarrow 1}_{\text{KR}, A}\) and \(\text{Exp}^{kr \leftarrow 0}_{\text{KR}, A}\); let \((N, e, S_1), (N, e, S_2), \ldots, (N, e, S_w)\) denote the responses to A’s wind oracle queries when A is in the member stage, \(w' \in \{0, 1, \ldots, q\}\). Let \(E_0\) and \(E_1\) respectively be the events in \(\text{Exp}^{kr \leftarrow 1}_{\text{KR}, A}\) and \(\text{Exp}^{kr \leftarrow 0}_{\text{KR}, A}\) that A queries its random oracle with a value \(S' \equiv S_{w'} \mod N\). We claim that

\[ \text{Adv}^{kr}_{\text{KR}, A} = \Pr \left[ \text{Exp}^{kr \leftarrow 1}_{\text{KR}, A} = 1 \land E_1 \right] - \Pr \left[ \text{Exp}^{kr \leftarrow 0}_{\text{KR}, A} = 1 \land E_0 \right]. \quad (18) \]

Consider now the adversary B in Figure 5. We additionally claim that

\[ \Pr \left[ \text{Exp}^{kr \leftarrow 1}_{\text{KR}, A} = 1 \land E_1 \right] \leq (q + 1) \cdot \Pr \left[ \text{Exp}^{rsa\text{-ow}}_{\text{KRsa}, B} = 1 \right]. \quad (19) \]
**Figure 5:** The adversary $B$ in the proof of Lemma 10.2.

Combining these two equations and the definition of security for $\mathcal{K}_{rsa}$ gives Equation (17).

It remains to justify Equation (18), Equation (19), and the resource requirements for $B$. We first justify Equation (18). Let $\Pr_1[\cdot]$ and $\Pr_0[\cdot]$ denote the probabilities over $\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1}$ and $\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0}$, respectively. From Definition 4.1, we have

$$\text{Adv}_{\mathcal{K}_{rsa},A}^{kr} = \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1} = 1] - \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0} = 1]$$

$$= \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1} = 1 \wedge \overline{E}_1] + \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1} = 1 \wedge \mathcal{E}_1]$$

$$- \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0} = 1 \wedge \overline{E}_0] - \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0} = 1 \wedge \mathcal{E}_0].$$

(20)

By conditioning,

$$\Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1} = 1 \wedge \overline{E}_1] = \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1} = 1 \mid \overline{E}_1] \cdot \Pr_1[\overline{E}_1]$$

and

$$\Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0} = 1 \wedge \overline{E}_0] = \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0} = 1 \mid \overline{E}_0] \cdot \Pr_0[\overline{E}_0].$$

Prior to $\mathcal{E}_1$ and $\mathcal{E}_0$, $A$’s view is identical in both experiments, meaning that

$$\Pr_1[\overline{E}_1] = \Pr_0[\overline{E}_0].$$

Further, if the events do not occur, then the outcome of $\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1}$ and $\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0}$ will be the same since the output of a random oracle is random if the input is unknown; i.e., the response to $A$’s key derivation oracle query in the non-member stage will be random in both cases and therefore

$$\Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1} = 1 \mid \overline{E}_1] = \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0} = 1 \mid \overline{E}_0].$$

Consequently,

$$\Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-1} = 1 \wedge \overline{E}_1] = \Pr[\text{Exp}_{\mathcal{K}_{rsa},A}^{kr-0} = 1 \wedge \overline{E}_0].$$

26
Combining the above equation with Equation (20) gives Equation (18).

We now turn to Equation (19). Note that \(B\) runs \(A\) exactly as in \(\text{Exp}_{K_{\text{rsa}},A}^{kr-1}\) assuming that \(B\) correctly guesses the number of wind oracle queries that \(A\) will make in its member stage; i.e., if \(B\) does not set \text{bad} to \text{true}. Here we use the fact that RSA encryption and decryption is a permutation and therefore \(B\) is justified in unwinding a starting state from its input \((y, e, N)\). Also observe that if \(\mathcal{E}_1\) in \(\text{Exp}_{K_{\text{rsa}},A}^{kr-1}\) occurs and if \(B\) does not set \text{bad} to \text{true}, then \(B\) will succeed in inverting RSA. Letting \(\text{BAD}\) denote the event that \(B\) sets \text{bad} to \text{true}, it follows that

\[
\Pr\left[\text{Exp}_{K_{\text{rsa}},A}^{kr-1} = 1 \land \mathcal{E}_1\right] \leq \Pr\left[\text{Exp}_{K_{\text{rsa}},B}^{\text{rsa-ow}} = 1 \mid \text{BAD}\right]
\]

and, by conditioning, that

\[
\Pr\left[\text{Exp}_{K_{\text{rsa}},A}^{kr-1} = 1 \land \mathcal{E}_1\right] \leq \Pr\left[\text{Exp}_{K_{\text{rsa}},B}^{\text{rsa-ow}} = 1 \land \text{BAD}\right] \cdot \frac{1}{\Pr_2[\text{BAD}]}
\]

where \(\Pr_2[\cdot]\) denotes the probability over \(\text{Exp}_{K_{\text{rsa}},B}^{\text{rsa-ow}}\). Equation (19) follows from the above equation and the fact that \(\Pr_2[\text{BAD}] = 1/(q + 1)\).

Turning to the resource requirements of \(B\), note that the for loop in \(B\) is not present in \(A\) (nor in the algorithm setup nor the experiment \(\text{Exp}_{K_{\text{rsa}},A}^{kr-b}\)). This means that \(B\) may perform \(q\) more RSA encryption operations than in the \(\text{Exp}_{K_{\text{rsa}},A}^{kr-b}\) experiment running \(A\); \(B\) does not, however, invoke any RSA decryption operations.

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.

**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 \(K_{\text{R}}\) be a key regression scheme. Given an adversary \(A\) attacking \(K_{\text{R}}\) that makes at most \(q_1\) queries to WindO and \(q_2\) queries to KeyderO, we can construct an adversary \(B\) attacking \(K_{\text{R}}\) such that

\[
\text{Adv}_{K_{\text{R}},A}^{kr} \leq q_2 \cdot \text{Adv}_{K_{\text{R}},B}^{kr}
\]

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

**Proof of Lemma 10.3:** We consider the case where \(q_2 = 0\) separately. If \(q_2 = 0\) then

\[
\Pr\left[\text{Exp}_{K_{\text{R}},A}^{kr-1} = 1\right] = \Pr\left[\text{Exp}_{K_{\text{R}},A}^{kr-0} = 1\right]
\]

since the adversary \(A\)’s view in the experiments \(\text{Exp}_{K_{\text{R}},A}^{kr-1}\) and \(\text{Exp}_{K_{\text{R}},A}^{kr-0}\) is identical. Therefore, when \(q_2 = 0\),

\[
\text{Adv}_{K_{\text{R}},A}^{kr} = \Pr\left[\text{Exp}_{K_{\text{R}},A}^{kr-1} = 1\right] - \Pr\left[\text{Exp}_{K_{\text{R}},A}^{kr-0} = 1\right] = 0 = q_2 \cdot \text{Adv}_{K_{\text{R}},B}^{kr}
\]
Experiment \( \text{ExpH}_{KR,A,i} \):

- Pick random oracle \( H \)
- \( l \leftarrow 0 \)
- \( \text{stp} \leftarrow \text{setup}^H \)
- \( \text{st} \leftarrow A^{\text{HWindO},H}(\text{member}) \)
- \( j \leftarrow 0 \)
- \( b' \leftarrow A^{\text{HKeyderO},i}(\text{non-member},\text{st}) \)
- Return \( b' \)

Oracle \( \text{HWindO} \):

- \( l \leftarrow l + 1 \)
- If \( l > MW \) then return \( \bot \)
- (\( \text{stp}, \text{stm} \)) \( \leftarrow \text{wind}^H(\text{stp}) \)
- Return \( \text{stm} \)

Oracle \( \text{HKeyderO},i \):

- \( l \leftarrow l + 1 \)
- If \( l > MW \) then return \( \bot \)
- (\( \text{stp}, \text{stm} \)) \( \leftarrow \text{wind}^H(\text{stp}) \)
- Else \( K \leftarrow \text{keyder}^H(\text{stm}) \)
- Return \( K \)

Figure 6: Hybrid experiments for the proof of Lemma 10.3.

for all adversaries \( B \).

We now restrict our analysis to the case where \( q_2 \geq 1 \). Consider the experiments \( \text{ExpH}_{KR,A,i} \) in Figure 6, \( i \in \{0, \ldots, q_2\} \). When \( i = q_2 \), \( \text{ExpH}_{KR,A,i} \) uses \text{keyder} to reply to all of \( A \)'s \( \text{HKeyderO} \) oracle queries, which means that

\[
\Pr[\text{Exp}^{kr-1}_{KR,A} = 1] = \Pr[\text{ExpH}^i_{KR,A,q_2} = 1].
\]

On the other hand, when \( i = 0 \), \( \text{ExpH}^i_{KR,A,i} \) replies to all of \( A \)'s \( \text{HKeyderO} \) oracle queries with random values from \( \text{DK} \), which means that

\[
\Pr[\text{Exp}^{kr-0}_{KR,A} = 1] = \Pr[\text{ExpH}^i_{KR,A,0} = 1].
\]

From these two equations we conclude that

\[
\text{Adv}^{kr}_{KR,A} = \Pr[\text{ExpH}^{i}_{KR,A,q_2} = 1] - \Pr[\text{ExpH}^{i}_{KR,A,0} = 1]. \tag{22}
\]

Consider now the adversary \( B \) in Figure 7. We claim that

\[
\Pr[\text{Exp}^{kr-1}_{KR,B} = 1] = \frac{1}{q_2} \sum_{i=0}^{q_2-1} \Pr[\text{ExpH}^{i}_{KR,A,i+1} = 1] \tag{23}
\]

and

\[
\Pr[\text{Exp}^{kr-0}_{KR,B} = 1] = \frac{1}{q_2} \sum_{i=0}^{q_2-1} \Pr[\text{ExpH}^{i}_{KR,A,i} = 1]. \tag{24}
\]

Subtracting Equation (24) from Equation (23) and using Definition 4.1, we get

\[
\text{Adv}^{kr}_{KR,B} = \Pr[\text{Exp}^{kr-1}_{KR,B} = 1] - \Pr[\text{Exp}^{kr-0}_{KR,B} = 1] = \frac{1}{q_2} \cdot \left( \Pr[\text{ExpH}^{i}_{KR,A,q_2} = 1] - \Pr[\text{ExpH}^{i}_{KR,A,0} = 1] \right). \tag{25}
\]

Equation (21) follows from combining Equation (22) with Equation (25).
Adversary $B^{\text{WindO}, H}(\text{member})$

```plaintext
i \sim \{0, \ldots, q_2 - 1\}
l \sim 0

Run $A^{\text{WindO}', H'}(\text{member})$, replying to $A$'s oracle queries as follows:

For each query to $\text{WindO}'$ do

\begin{align*}
stm &\sim \text{WindO} \\
l &\sim l + 1 \\
\text{If } l > MW &\text{ then } stm \sim \perp \\
\text{Return } stm &\text{ to } A \\
\text{For each query } x &\text{ to } H' \text{ do} \\
y &\sim H(x) \\
\text{Return } y &\text{ to } A \\
\text{Until } A \text{ halts outputting a state } st'
\end{align*}

For $j = 0$ to $i - 1$ do

\begin{align*}
stm &\sim \text{WindO} \\
K_j &\sim \text{keyder}^H(stm) \\
st &\sim (st', i, l, K_0, \ldots, K_{i-1}) \\
\text{Return } st
\end{align*}
```

Adversary $B^{\text{KeyderO}_b, H}(\text{non-member, st})$

Parse $st$ as $(st', i, l, K_0, \ldots, K_{i-1})$

```plaintext
j \sim 0

Run $A^{\text{KeyderO}', H'}(\text{non-member, st}')$, replying to $A$'s oracle queries as follows:

For each query to $\text{KeyderO}'$ do

\begin{align*}
\text{If } j < i &\text{ then } K \sim K_j \\
\text{Else if } j = i &\text{ then } K \sim \text{KeyderO}_b \\
\text{Else } K &\sim DK \\
j &\sim j + 1 \\
l &\sim l + 1 \\
\text{If } l > MW &\text{ then } K \sim \perp \\
\text{Return } K &\text{ to } A \\
\text{For each query } x &\text{ to } H' \text{ do} \\
y &\sim H(x) \\
\text{Return } y &\text{ to } A \\
\text{Until } A \text{ halts outputting a bit } b \\
\text{Return } b
\end{align*}
```

Figure 7: Adversary $B$ for the proof of Lemma 10.3. We describe in the body an alternate description with reduced resource requirements.

---

It remains to justify Equation (23), Equation (24), and the resources of $B$. To justify Equation (23), note that in the experiment $\text{Exp}_{KR,B}^{kr-1}$, when $B$ picks some value for $i$, the view of $A$ becomes equivalent to $A$'s view in $\text{Exp}_{KR,A,i+1}$; namely, $A$'s first $i + 1$ queries to its $\text{KeyderO}$ oracle will be computed using $\text{keyder}$, and the remaining $\text{KeyderO}$ oracle queries will return random values from $\text{DK}$. More formally, if $I$ denotes the random variable for the $B$'s selection for the variable $i \in \{0, \ldots, q_2 - 1\}$, then

$$\text{Pr} \left[ \text{Exp}_{KR,B}^{kr-1} = 1 \mid I = i \right] = \text{Pr} \left[ \text{Exp}_{KR,A,i+1}^{kr-1} = 1 \right]$$

for each $i \in \{0, \ldots, q_2 - 1\}$. Letting $\text{Pr}_1[\cdot]$ denote the probability over $\text{Exp}_{KR,B}^{kr-1}$, we then derive Equation (23) by conditioning off the choice of $i$:

$$\text{Pr} \left[ \text{Exp}_{KR,B}^{kr-1} = 1 \right] = \sum_{i=0}^{q_2-1} \text{Pr} \left[ \text{Exp}_{KR,B}^{kr-1} = 1 \mid I = i \right] \cdot \text{Pr}_1[ I = i ]$$

$$= \frac{1}{q_2} \cdot \sum_{i=0}^{q_2-1} \text{Pr} \left[ \text{Exp}_{KR,B}^{kr-1} = 1 \mid I = i \right]$$

$$= \frac{1}{q_2} \cdot \sum_{i=0}^{q_2-1} \text{Pr} \left[ \text{Exp}_{KR,A,i+1}^{kr-1} = 1 \right]$$

The justification for Equation (24) is similar. When $B$ picks some value for $i$ in $\text{Exp}_{KR,B}^{kr-0}$, the view of $A$ in $\text{Exp}_{KR,B}^{kr-0}$ becomes equivalent to $A$'s view in $\text{Exp}_{KR,A,i}$ since in both cases the responses
to $A$’s first $i$ (not $i + 1$ this time) queries to its KeyderO oracle will be computed using keyder, and the remaining KeyderO oracle queries will return random values from DK.

We now turn to the resource requirements of $B$. The pseudocode for $B$ in Figure 7 suggests that $B$ might invoke WindO and keyder up to $q_2$ times more than $A$ (since the last for loop of $B$’s member stage runs for up to $q_2$ interactions even though $A$ may not make that many KeyderO oracle queries). We describe $B$ this way since we feel that Figure 7 better captures the main idea behind our proof and what $B$ does. Equivalently, $B$ could split $A$’s non-member stage between its (B’s) own member and non-member stages and invoke WindO and keyder only the number of times that it needs to simulate $i$ of $A$’s KeyderO$H$ oracle queries. When viewed this way, $B$ uses resources equivalent, within a constant factor, to the resources of $A$.

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

$$\text{Adv}^{kr}_{KR,A} \leq 2q^2 \cdot \text{Adv}^{rsa-ow}_{K_{rsa},B},$$

where $q$ is the maximum number of winding and key derivation oracle queries that $A$ makes. Adversary $B$ uses resources within a constant factor of the resources of $A$ 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 $B$ 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 [23] 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 [23] 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 [24] to provide access control. Chefs uses lazy revocation [22, 33] 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 1024-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.

Microbenchmarks of Chefs in a static environment ensure confidence that the basic file system performs in a reasonable manner. The microbenchmarks help to explain the consumption of time in Chefs. Application-level measurements show that a publisher can serve many keys per second to clients when using key regression — even on a low-bandwidth connection.

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 7200 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 [49] measured the round-trip latency between the pairs of machines at 249 \( \mu \)sec, 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 69632 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 [45] 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 16384 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 9 indicate minimum and maximum measurements.
Table 2: Small-file, large-file, and emacs-compilation microbenchmarks of Chefs versus SFSRO, an integrity-protected but not access-controlled file system. In all tests the server has a warm cache, and the client has a cold cache.

<table>
<thead>
<tr>
<th></th>
<th>Small file</th>
<th>Large file</th>
<th>Emacs compilation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFSRO</td>
<td>1.38 sec</td>
<td>8.21 Mbyte/sec</td>
<td>50.78 sec</td>
</tr>
<tr>
<td>Chefs</td>
<td>1.52 sec</td>
<td>5.13 Mbyte/sec</td>
<td>51.43 sec</td>
</tr>
</tbody>
</table>

11.2 Microbenchmarks

To isolate the cost of encryption in Chefs, we disable key regression and compare the performance of Chefs to SFSRO using I/O-bound workloads. In these workloads, the Chefs client performs a single key derivation but no unwinding. SFSRO is an integrity-protected, but not access-controlled file system. Because SFSRO has no access control, there is no encryption. These measurements can later help to explain where time is spent in application-level benchmarks.

Table 2 analyzes the performance of Chefs with the small-file, large-file, and emacs-compilation benchmarks described in the SFSRO paper [24]. The small-file benchmark consists of the read phases of the LFS benchmarks [46] — sequentially reading 1000 files each 1 Kbyte in size spread across ten directories. The small-file benchmark helps to understand the performance of a single client accessing a single server. The large-file benchmark generates a sequential read of a 40 Mbyte file. Using a warm server cache and cold client cache allows the large-file benchmark to measure the overhead added to a single client’s throughput. The emacs-compilation benchmark measures the execution time on a client to compile Emacs. The source code is fetched from a Chefs or SFSRO server, and the object code is compiled locally on the client. Optimization and debugging are disabled during the compilation.

The SFSRO and Chefs small-file benchmarks each generate 2022 RPCs to fetch and decrypt content from a server (1000 files, 10 directories, and one root directory — each generating two RPCs: one for the inode, one for the content). The measurements below show that there is a performance cost to adding confidentiality in Chefs for I/O-intensive workloads, but that the cost is not noticeable in application-level benchmarks.

The SFSRO and Chefs small file benchmarks finish in 1.38 seconds and 1.52 seconds respectively. The small overhead in Chefs comes as a result of decrypting content with 128-bit AES in CBC mode, downloading a 20-byte key regression member state from the publisher, and decrypting the member state with 1024-bit RSA. In this local area network, the network latency accounts for nearly 30% of the overall latency; 2022 RPCs with a 249 µsec round-trip time yields 503 msec. Content distribution networks are more commonly found in wide-area networks, where longer round-trip times would absorb the cost of the cryptography in Chefs.

The large-file benchmark generates 5124 RPCs to fetch 40 Mbytes of content from the server (two RPCs for the root directory, two for the file, and 5120 for the file content). The cost of cryptography in Chefs comes at a cost of 3.08 MByte/sec in throughput. The Chefs client takes approximately 32% of the client CPU, whereas the SFSRO client takes only 14% of the CPU.

The software distribution benchmark consists of an Emacs version 21.3 compilation. The source code is stored in the file system, while the resulting binaries are written to a local disk. The experiment mounts the remote file system, runs configure, then compiles with make. This CPU-intensive workload requires access to approximately 300 files. The cost of cryptography is no longer noticeable. The Chefs client program consumes less than 1% of the CPU while the compiler takes nearly 90% of the CPU.
Table 3: Microbenchmarks of KR-SHA1, KR-AES, KR-RSA key regression.

<table>
<thead>
<tr>
<th>Key regression protocol</th>
<th>Winds/sec</th>
<th>Unwinds/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>KR-SHA1</td>
<td>Not applicable</td>
<td>687 720</td>
</tr>
<tr>
<td>KR-AES</td>
<td>Not applicable</td>
<td>3 303 900</td>
</tr>
<tr>
<td>KR-RSA</td>
<td>158</td>
<td>35 236</td>
</tr>
</tbody>
</table>

11.3 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 3 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 [47]. 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, 13, 27, 29, 50, 53], 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 1024-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 8 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.
Figure 8: 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.

The benchmark measured in Figure 8 effectively simulates the effect of 20 clients applying the same key distribution workload to the server. After all traces have completed, the effective number of trace playbacks per second is recorded. Each test runs for 1–2 seconds, asynchronously playing back 20 traces of a single client fetching the keys for the search workload.

A Chefs trace consists of a TCP connection setup, followed by a `getkey` RPC. Chefs always generates a single `getkey` remote procedure call, regardless of the number of key versions.

A Sous-Chefs trace consists of a TCP connection setup, followed by a read of an encrypted file containing a set of keys. The file read is further composed of an `sfsconnect` RPC, a `getfsinfo` RPC, a `getkey` RPC, and a number of `getdata` RPCs sufficient to download the file of keys. The Sous-Chefs traces of fetching $1, 10, 10^2, 10^3, 10^4, 10^5,$ and $10^6$ keys generate $4, 4, 4, 5, 24, 200,$ and $1966$ asynchronous RPCs respectively.

Over fast network connections, the cost of transferring the 10,000 8 KB files dominates the client latency. A new trend appears after 100 transferred keys in the measurements of Sous-Chefs in Figure 8. The network bandwidth and latency of the publisher begin to dominate the client latency. For instance, Sous-Chefs running on the simulated cable modem with 100 keys results in a publisher having 13 client-sessions/sec. This measurement meets the expectations. With a 384 Kbit/sec upload bandwidth, 20 ms round-trip delay, and the transfer of 100 keys each of size 16 bytes using 4 RPCs, one would expect a single client to take at least 50 msec simply to download the keys. This translates to at most 20 client-sessions/sec under perfectly asynchronous RPC conditions—confirming the measurements as reasonable.

**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 9 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
Figure 9: A log-log chart of single client latency to read 10000 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 by the transfer time of keys after reaching 10000 keys. Key regression enables much better latency in Chefs.

measurement for Sous-Chefs downloading 1 000 000 keys because the kernel assumes that the mount failed after waiting 1000 seconds. On a 56 Kbit/sec network, Sous-Chefs is expected to take over 2 232 seconds to download 1000 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.

Acknowledgments

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


