Manuscript.

# Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage

Kevin Fu[*] Seny Kamara[†] Tadayoshi Kohno[‡]

September 3, 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. 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, EBU3B, Room 4240, 9500 Gilman Drive, La Jolla, CA 92093-0404, USA. E-mail: `tkohno@cs.ucsd.edu`. URL: `http://www-cse.ucsd.edu/users/tkohno`. Support by an IBM Ph.D. Fellowship and NSF grant CCR 0093337. Part of this research was performed while visiting the University of California at Berkeley.

# 1 Introduction

Content distribution networks (CDNs) such as Akamai [2], BitTorrent [10], and Coral [15] 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 [17, 19, 22, 23, 26, 28]. Under the *lazy revocation* model for access control [17, 23], 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 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 [23] 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 [23], the critical properties of a key rotation scheme are that (1) given the $i$-th key $K_i$ it is easy to compute the keys $K_j$ for all previous time periods $j < i$, but (2) for any time period $l > i$ after $i$, it should be computationally infeasible to compute the keys $K_l$ for time period $l$ given only $K_i$. 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 key regression in a secure file system and we analyze the performance of key regression within the context of this file system. We summarize our contributions in more detail in the following paragraphs.

**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* subsequent keys $K_l$, $l > i$, the evicted member *can distinguish* subsequent keys $K_l$ from random. The lack of pseudorandomness follows from the fact that if an evicted member is given the real key $K_l$, then by definition (i.e., by property (2)) the evicted member can recover the real key $K_i$; but given a random key instead of $K_l$, the evicted member will with high probability recover a key $K_i' \neq K_i$. The difference between unpredictability and lack of pseudorandomness can have severe consequences in practice. To illustrate the seriousness of this design flaw, we describe a key rotation scheme and a symmetric encryption scheme that individually meet their desired security properties (property (2) for key rotation and IND-CPA privacy for symmetric encryption [3]), 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

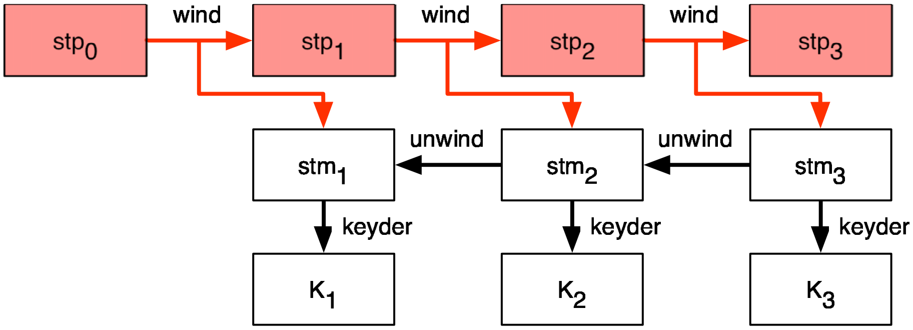

Figure 1: Key regression overview; $\mathsf{stp}_i$ and $\mathsf{stm}_i$ respectively represent the $i$-th publisher and member states.

fails to provide even a weak form of privacy.[1]

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

To describe key regression, we must enact a paradigm shift: rather than give a new member the $i$-th key $K_i$ directly, the content publisher would give the member a *member state* $\mathsf{stm}_i$. 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 $\mathsf{stm}_j$, $j < i$. 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 $\mathsf{stm}_i$. 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 $\mathsf{stm}_i$, it is still computationally infeasible for the evicted member to compute the $l$-th member state $\mathsf{stm}_l$. Viewed another way, there is no path from $K_l$ to $\mathsf{stm}_i$ in Figure 1 and vice-versa.

**Our constructions.** We refer to our three preferred key regression schemes as KR-RSA, KR-SHA1, and KR-AES. Rather than rely solely on potentially error-prone heuristic methods for analyzing the security of our constructions, we prove that all three are secure key regression schemes. Our security proofs use the reduction-based provable security approach pioneered by Goldwasser and Micali [20] and lifted to the concrete setting by Bellare, Kilian, and Rogaway [4]. 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 [5]. For the proof of KR-AES, we assume that AES is a secure pseudorandom permutation [25, 4].

**Implementation and evaluation.** We integrated key regression in 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

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

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 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.

We also provide the first empirical measurements of either a key regression or key rotation scheme. Contrary to conventional wisdom, we find that KR-AES can perform more than four times as many unwinds/sec than KR-SHA1. The measurements will help developers select the most appropriate key regression scheme for particular applications.

**Applications.** We target key regression at 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, startup, or 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 [2]. And it controls access to these distributions by only releasing the `.torrent` files to its members. Using key regression, 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 time period $i$.[3]

### 1.2 Related work

The key rotation scheme in Plutus [23] inspired our research in key regression. Bellare and Yee [7] 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 $\mathsf{stm}_i$ from distinguishing *subsequent* encryption keys $K_l$, $l > i$, from random, a FSPRG is designed to prevent an adversary who learns the state of the FSPRG at some point in time from distinguishing *previous* outputs of the FSPRG from random. In our security proof for KR-AES, we make the relationship between key regression and FSPRGs concrete by first proving that one can build a secure key regression scheme from any secure FSPRG by essentially running the FSPRG backwards. Abdalla and Bellare [1] formally analyze methods for rekeying symmetric encryption schemes, but like FSPRGs, a standard rekeying scheme is roughly the mirror image of a key regression scheme.

As pointed out in [9], one possible mechanism for distributing updated content encryption keys for a secure file system is to use a broadcast encryption scheme [14, 27, 12, 13]. Indeed, one of the main challenges faced by an encrypted file system is the distribution of the encryption keys to

[2] `http://club.mandriva.com/xwiki/bin/view/Downloads/TorrentAccess`.

[3] While Mandriva may wish to exercise access control over non-security-critical patches and upgrades, they would likely wish to allow all Mandriva users, including evicted Mandriva Club members, access to all security-critical patches. To enable this, 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.

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

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

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

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

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

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

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

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

the remaining (not evicted) set of users, and broadcast encryption provides an ideal solution. We note, however, that this is orthogonal to the specific problem addressed by key regression. In fact, 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 [33] protocols are resilient even when broadcasts are lost on the network. In this manner, one can view key regression as having the self-healing property in perpetuity.

## 2 Notation.

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

We use $\mathrm{AES}_K(M)$ to denote the process of running the AES block cipher with key $K$ and input block $M$. An RSA [29] key generator for some security parameter $k$ is a randomized algorithm $\mathcal{K}_{\rm rsa}$ that returns a triple $(N, e, d)$. The modulus $N$ is the product of two distinct odd primes $p, q$ such that $2^{k-1} \leq N < 2^k$; the encryption exponent $e \in \mathbb{Z}^*_{\varphi(N)}$ and the decryption exponent $d \in \mathbb{Z}^*_{\varphi(N)}$ are such that $ed \equiv 1 \bmod \varphi(N)$, where $\varphi(N) = (p-1)(q-1)$. Section 5 describes what it means for an RSA key generator to be one-way.

## 3 Problems with key rotation

A key rotation scheme [23] consists of three algorithms: setup, wndkey, and unwndkey. Figure 2 shows the original (RSA-based) Plutus key rotation scheme. Following Plutus, one familiar with hash chains [24] and S/KEY [21] might design the key rotation scheme in Figure 3, which is more efficient than the scheme in Figure 2, but which is limited because it can only produce $\mathsf{MW}$ ("max wind") keys, where $\mathsf{MW}$ is a parameter chosen by the implementor. A content publisher runs the $\mathsf{setup}$ algorithm to initialize a key rotation scheme; the result is public information $\mathsf{pk}$ for all users and a secret $\mathsf{sk}_1$ for the content publisher. The content publisher invokes $\mathsf{wndkey}(\mathsf{sk}_i)$ to obtain the key $K_i$ and a new secret $\mathsf{sk}_{i+1}$. Any user in possession of $K_i$, $i > 1$, and $\mathsf{pk}$ can invoke $\mathsf{unwndkey}(K_i, \mathsf{pk})$ to obtain $K_{i-1}$. Informally, the desired security property of a key rotation scheme is that, given only $K_i$ and $\mathsf{pk}$, it should be computationally infeasible for an evicted member (the adversary) to compute $K_l$, for any $l > i$. The Plutus construction in Figure 2 has this property under the RSA one-wayness assumption (defined in Section 5), and the construction in Figure 3 has this property if one replaces SHA1 with a random oracle [5].

**The problem.** In Section 1 we observed that the $l$-th key output by a key rotation scheme cannot be pseudorandom, i.e., will be distinguishable from a random string, to an ex-member in possession of the key $K_i$ for some previous time period $i < l$.[4] 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 boxes.

For our example, we first present a key rotation scheme $\overline{\mathcal{KO}}$ and an encryption scheme $\overline{\mathcal{SE}}$ that individually both satisfy their respective security goals (unpredictability for the key rotation scheme and IND-CPA privacy [3] for the symmetric encryption scheme). To build $\overline{\mathcal{KO}}$, we start with a secure key rotation scheme $\mathcal{KO}$; $\overline{\mathcal{KO}}$ outputs keys twice as long as $\mathcal{KO}$. The $\overline{\mathcal{KO}}$ winding algorithm $\overline{\mathsf{wndkey}}$ invokes $\mathcal{KO}$'s winding algorithm to obtain a key $K$; $\overline{\mathsf{wndkey}}$ then returns $K \| K$ as its key. On input a key $K \| K$, $\overline{\mathsf{unwndkey}}$ invokes $\mathcal{KO}$'s unwinding algorithm with input $K$ to obtain a key $K'$; $\overline{\mathsf{unwndkey}}$ then returns $K' \| K'$ as its key. If the keys output by $\mathsf{wndkey}$ are unpredictable to evicted members, then so must the keys output by $\overline{\mathsf{wndkey}}$. To build $\overline{\mathcal{SE}}$, we start with a secure symmetric encryption scheme $\mathcal{SE}$; $\overline{\mathcal{SE}}$ uses keys that are twice as long as $\mathcal{SE}$. The $\overline{\mathcal{SE}}$ encryption and decryption algorithms take the key $K$, split it into two halves $K = L_1 \| L_2$, and run $\mathcal{SE}$ with key $L_1 \oplus L_2$. If the key $K$ is random, then the key $L_1 \oplus L_2$ is random and $\overline{\mathcal{SE}}$ runs the $\mathcal{SE}$ encryption algorithm with a uniformly selected random key. This means that $\overline{\mathcal{SE}}$ satisfies the standard IND-CPA security goal if $\mathcal{SE}$ does.

Despite the individual security of both $\overline{\mathcal{KO}}$ and $\overline{\mathcal{SE}}$, when the keys output by $\overline{\mathcal{KO}}$ are used to key $\overline{\mathcal{SE}}$, $\overline{\mathcal{SE}}$ will always run $\mathcal{SE}$ with the all-zero key; i.e., the content publisher will encrypt all content under the same constant key. An adversary can thus trivially compromise the privacy of all encrypted data, including data encrypted during time periods $l > i$ after being evicted. Although the construction of $\overline{\mathcal{KO}}$ and $\overline{\mathcal{SE}}$ may seem somewhat contrived, 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

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

adversary evicted at time $i$. Here we formalize such an object: a key regression scheme. Following the reduction-based practice-oriented provable security approach [20, 4], our formalisms involve carefully defining the syntax, correctness requirements, and security goal of a key regression scheme. These formalisms enable us to, in Section 5, prove that our preferred constructions are secure under reasonable assumptions. We desire provable security over solely *ad hoc* analyses since, under *ad hoc* methods alone, one can never be completely convinced that a cryptographic construction is secure even if one assumes that the underlying components (e.g., block ciphers, hash functions, RSA) are secure.

**Overview of key regression.** Figure 1 gives an abstract overview of a key regression scheme. The content publisher has content publisher states $\mathsf{stp}_i$ 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 $\mathsf{stm}_i$. As the arrows in Figure 1 suggest, given $\mathsf{stm}_i$, 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 $\mathsf{stm}_l$, $l > i$, 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 $\mathsf{stm}_i$, 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 SHA1($K_i$), instead. If one models SHA1 as a random oracle and if the key rotation scheme produces unpredictable future keys $K_l$, then it might seem reasonable to conclude that an ex-member given $K_i$ should not be able to distinguish future values SHA1($K_l$), $l > i$, from random. While this reasoning may be sound for some specific key rotation schemes (this reasoning actually serves as the basis for our derivative of the construction in Figure 2, KR-RSA in Construction 5.5) 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($K_l$) may not work with all key rotation schemes, which means that the engineer cannot use a key rotation scheme as a black box. For example, while SHA1($K_l$) would work for the scheme in Figure 2, it would cause problems for the scheme in Figure 3. We choose to consider a new cryptographic object, key regression, because we desire a cryptographic object that is not as prone to accidental misuse. Additionally, by focusing attention on a new cryptographic object, we allow ourselves greater flexibility in how we can construct objects that meet our requirements. For example, one of our constructions (KR-AES, Construction 5.3) does not use a hash function and is therefore in the standard model instead of the random oracle model.

## 4.1 Syntax and correctness requirements

**Syntax.** Here we formally define the syntax of a key regression scheme $\mathcal{KR} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$. Let $H$ be a random oracle; all four algorithms are given access to the random oracle, though they may not use the random oracle in their computations. Via $\mathsf{stp} \stackrel{\$}{\leftarrow} \mathsf{setup}^H$, the randomized setup algorithm returns a publisher state. Via $(\mathsf{stp}', \mathsf{stm}) \stackrel{\$}{\leftarrow} \mathsf{wind}^H(\mathsf{stp})$, the randomized winding algorithm takes a publisher state $\mathsf{stp}$ and returns a pair of publisher and member states or the error code $(\perp, \perp)$. Via $\mathsf{stm}' \leftarrow \mathsf{unwind}^H(\mathsf{stm})$ the deterministic unwinding algorithm takes a member state

$\mathsf{stm}$ and returns a member state or the error code $\perp$. Via $K \stackrel{\$}{\leftarrow} \mathsf{keyder}^H(\mathsf{stm})$ the deterministic key derivation algorithm takes a member state $\mathsf{stm}$ and returns a key $K \in \mathsf{DK}$, where $\mathsf{DK}$ is the *derived key space* for $\mathcal{KR}$. Let $\mathsf{MW} \in \{1, 2, \ldots\} \cup \{\infty\}$ denote the maximum number of derived keys that $\mathcal{KR}$ is designed to produce. We do not define the behavior of the algorithms when input the error code $\perp$. A construction may use multiple random oracles, but since one can always obtain multiple random oracles from a single random oracle [5], our definitions assume just one.

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

**Remarks on syntax.** Although we allow $\mathsf{wind}$ to be randomized, the $\mathsf{wind}$ algorithms in all of our constructions are deterministic. We allow $\mathsf{wind}$ to return $(\perp, \perp)$ since we only require that $\mathsf{wind}$ return non-error states for its first $\mathsf{MW}$ invocations. We use the pair $(\perp, \perp)$, rather than simply $\perp$, to denote an error from $\mathsf{wind}$ since doing so makes our pseudocode cleaner. We allow $\mathsf{unwind}$ to return $\perp$ since the behavior of $\mathsf{unwind}$ may be undefined when input the first member state.

### 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 $\mathsf{stm}_l$, $l > i$, 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 $\mathsf{WindO}$ oracle). Note that the $\mathsf{WindO}$ 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 $\mathsf{KeyderO}$. The key derivation oracle will either return real derived keys (via internal calls to $\mathsf{wind}$ and $\mathsf{keyder}$) or random keys. The adversary's goal is to guess whether the $\mathsf{KeyderO}$ oracle's responses are real derived keys or random keys. Since the publisher is in charge of winding and will not invoke the winding algorithm more than the prescribed maximum number of times, $\mathsf{MW}$, the $\mathsf{WindO}$ and $\mathsf{KeyderO}$ oracles in our security definition will only respond to the first $\mathsf{MW}$ queries from the adversary.

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

```
**Experiment** $\mathbf{Exp}^{\mathrm{kr}\text{-}b}_{\mathcal{KR},\mathcal{A}}$
    Pick random oracle $H$
    $i \leftarrow 0$
    $\mathsf{stp} \stackrel{\$}{\leftarrow} \mathsf{setup}^H$
    $\mathsf{st} \stackrel{\$}{\leftarrow} \mathcal{A}^{\mathsf{WindO},H}(\mathsf{member})$
    $b' \stackrel{\$}{\leftarrow} \mathcal{A}^{\mathsf{KeyderO}_b,H}(\mathsf{non\text{-}member}, \mathsf{st})$
    Return $b'$
```

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

```
**Oracle** $\mathsf{KeyderO}_b$
    $i \leftarrow i + 1$
    If $i > \mathsf{MW}$ then return $\perp$
    $(\mathsf{stp}, \mathsf{stm}) \stackrel{\$}{\leftarrow} \mathsf{wind}^H(\mathsf{stp})$
    If $b = 1$ then
        $K \leftarrow \mathsf{keyder}^H(\mathsf{stm})$
    If $b = 0$ then
        $K \stackrel{\$}{\leftarrow} \mathsf{DK}$
    Return $K$
```

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

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

---

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

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

Under the concrete security approach [4], we say that a $\mathcal{KR}$ is "KR-secure" if for any adversary $\mathcal{A}$ attacking $\mathcal{KR}$ with resources (running time, size of code, number of oracle queries) limited to "practical" amounts, the KR-advantage of $\mathcal{A}$ is "small." Formal results are stated with concrete bounds. ▮

## 5 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 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).

We begin by defining KR-SHA1. 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.

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

```
Alg. setup
  $\mathsf{stm}_{\mathsf{MW}} \xleftarrow{\$} \{0,1\}^{160}$
  For $i = \mathsf{MW}$ downto 2 do
    $\mathsf{stm}_{i-1} \leftarrow \mathsf{unwind}(\mathsf{stm}_i)$
  $\mathsf{stp} \leftarrow \langle 1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}} \rangle$
  Return stp
```

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

```
Alg. unwind(stm)
  $\mathsf{stm}' \leftarrow \mathrm{SHA1}(\mathsf{stm})$
  Return $\mathsf{stm}'$

Alg. keyder(stm)
  $K \leftarrow \mathrm{SHA1}(0^8 \| \mathsf{stm})$
  Return $K$
```

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

The following theorem states that KR-SHA1 is secure in the random oracle model for adversaries that make a reasonable number of queries to their random oracles. Here we view the application of SHA1$(\cdot)$ in unwind as one random oracle and the application of SHA1$(0^8\|\cdot)$ in keyder as another random oracle. The proof of Theorem 5.2 in Appendix A is thus in the random oracle model [5].

**Theorem 5.2** *Let $\mathcal{KR}$ be a generalization of* KR-SHA1 *(Construction 5.1) in which SHA1$(\cdot)$ in* unwind *is replaced by a random oracle $H_1$: $\{0,1\}^{160} \to \{0,1\}^{160}$ and in which SHA1$(0^8\|\cdot)$ in* keyder *is replaced by another random oracle $H_2$: $\{0,1\}^{160} \to \{0,1\}^{160}$. Then $\mathcal{KR}$ is KR-secure in the random oracle model. Concretely, for any adversary $\mathcal{A}$ we have*

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

*where $q$ is the maximum number of queries that $\mathcal{A}$ makes to its random oracles.* ▮

Our next preferred construction, KR-AES, uses AES and is provably secure in the standard model.

**Construction 5.3** [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
  $\mathsf{stm}_{\mathsf{MW}} \xleftarrow{\$} \{0,1\}^{128}$
  For $i = \mathsf{MW}$ downto 2 do
    $\mathsf{stm}_{i-1} \leftarrow \mathsf{unwind}(\mathsf{stm}_i)$
  $\mathsf{stp} \leftarrow \langle 1, \mathsf{stm}_1, \ldots, \mathsf{stm}_{\mathsf{MW}} \rangle$
  Return stp
```

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

```
Alg. unwind(stm)
  $\mathsf{stm}' \leftarrow \mathrm{AES}_{\mathsf{stm}}(0^{128})$
  Return $\mathsf{stm}'$
Alg. keyder(stm)
  $K \leftarrow \mathrm{AES}_{\mathsf{stm}}(1^{128})$
  Return $K$
```

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

Before proving the security of KR-AES, we first recall the standard notion of a pseudorandom permutation [25, 4]. Let $E$: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher and let Perm$(l)$ denote the set of all permutations on $\{0,1\}^l$. If $\mathcal{A}$ is an adversary with access to an oracle, we let

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

denote the *prp-advantage* of $\mathcal{A}$ in attacking $E$. Under the concrete security approach [4], there is no formal definition of what it means for $E$ to be a "secure PRP," but in discussions this phrase should be taken to mean that, for any $\mathcal{A}$ attacking $E$ with resources (running time, size of code) limited to "practical" amounts, the prp-advantage of $\mathcal{A}$ is "small." The formal result below is stated with concrete bounds.

**Theorem 5.4** *If AES is a secure PRP, then* KR-AES *is KR-secure. Concretely, given an adversary $\mathcal{A}$ attacking* KR-AES*, we can construct an adversary $\mathcal{B}$ attacking AES such that*

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

*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.* ▮

The proof of Theorem 5.4 is in Appendix B. The proof exploits the fact that one can roughly view KR-AES as the mirror image (reverse) of one of Bellare and Yee's forward-secure pseudorandom bit generators [7].

Our final construction, KR-RSA derives from the key rotation scheme in Figure 2; KR-RSA differs from KR-SHA1 and KR-AES in that $\mathsf{MW} = \infty$, meaning that the content provider can invoke the KR-RSA winding algorithm an unbounded number of times.

**Construction 5.5** [KR-RSA] The key regression scheme KR-RSA = (setup, wind, unwind, keyder) is defined as follows. Let $\mathcal{K}_{\mathrm{rsa}}$ be an RSA key generator for some security parameter $k$ and let $m\colon \mathbb{Z}_{2^k} \to \{0,1\}^k$ denote the standard big-endian encoding of the integers in $\mathbb{Z}_{2^k}$ to $k$-bit strings.

```
Alg. setup
    $(N, e, d) \xleftarrow{\$} \mathcal{K}_{\mathrm{rsa}}$
    $S \xleftarrow{\$} \mathbb{Z}_N^*$
    $\mathsf{stp} \leftarrow \langle N, e, d, S \rangle$
    Return stp

Alg. wind(stp)
    Parse stp as $\langle N, e, d, S \rangle$
    $S' \leftarrow S^d \bmod N$
    $\mathsf{stp}' \leftarrow \langle N, e, d, S' \rangle$
    $\mathsf{stm} \leftarrow \langle N, e, S \rangle$
    Return $(\mathsf{stp}', \mathsf{stm})$

Alg. unwind(stm)
    Parse stm as $\langle N, e, S \rangle$
    $S' \leftarrow S^e \bmod N$ ; $\mathsf{stm}' \leftarrow \langle N, e, S' \rangle$
    Return $\mathsf{stm}'$

Alg. keyder(stm)
    Parse stm as $\langle N, e, S \rangle$ ; $K \leftarrow \mathrm{SHA1}(m(S))$
    Return $K$
```

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

Before presenting Theorem 5.6, we first recall the standard notion of one-wayness for RSA. Let $\mathcal{K}_{\mathrm{rsa}}$ be an RSA key generator with security parameter $k$. If $\mathcal{A}$ is an adversary, we let

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

denote the RSA one-way advantage of $\mathcal{A}$ in inverting RSA with the key generator $\mathcal{K}_{\mathrm{rsa}}$. Under the concrete security approach [4], there is no formal definition of what it means for $\mathcal{K}_{\mathrm{rsa}}$ to be "one-way." In discussions this phrase should be taken to mean that for any $\mathcal{A}$ attacking $\mathcal{K}_{\mathrm{rsa}}$ with resources (running time, size of code, number of oracle queries) limited to "practical" amounts, the RSA one-way advantage of $\mathcal{A}$ is "small." The formal result below is stated with concrete bounds. We prove Theorem 5.6 in Appendix C; the proof is in the random oracle model.

**Theorem 5.6** *Let $\mathcal{KR}$ be a generalization of* KR-RSA *in which SHA1$(m(\cdot))$ in* keyder *is replaced by a random oracle $H\colon \mathbb{Z}_{2^k} \to \{0,1\}^{160}$. If $\mathcal{K}_{rsa}$ is an RSA key generator with security parameter $k$, then $\mathcal{KR}$ is KR-secure under the RSA one-wayness assumption. Concretely, given an adversary $\mathcal{A}$ attacking* KR-RSA*, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{K}_{rsa}$ such that*

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

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

## 6 Performance of key regression in access-controlled content distribution

We integrated key regression into the Chefs file system [17] (described in a thesis) 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 [17] 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 [18, 23] to provide access control. Chefs uses lazy revocation [16] 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. The publisher may often be offline, effectively providing zero throughput.

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

## 6.1 Hypothesis and methodology

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

**Experimental setup.** We used three machines to benchmark key regression in Chefs. 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. With NetPipe [32] we 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 69,632 KB. When writing in 8 KB chunks (the block size in Chefs), the peak TCP throughput was 66 Mbit/sec.

To simulate cable modem and analog modem network conditions, we used the dummynet [30] driver in FreeBSD. For the cable modem, we set the round-trip delay to 20 msec and the download and upload bandwidth to 4 Mbit/sec and 384 Kbit/sec respectively. For the analog modem, we set the round-trip delay to 200 msec and the upload and download bandwidth each to 56 Kbit/sec.

For each measurement, we report the median result of five samples.

## 6.2 Secure content distribution on untrusted storage

We were not able to find a standard benchmark for measuring the effects of group membership dynamics. Therefore, we evaluate Chefs based on how a client might search for content in an online

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

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

newspaper. We benchmark the performance of Chefs based on this workload.

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

**A search workload.** Our benchmarks were inspired by the membership dynamics reported at Salon.com, a subscription-based online journal[5]. 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, we created a file system containing 10,000 8 KB encrypted files and the associated content keys. Our experiment consists of mounting the file system and reading all the files. This causes the client machine to fetch all the content keys.

Note that we cannot let the untrusted server perform the search because a client could not believe in the response. For instance, an untrusted server could respond, "No results found." Moreover, the server is not able to selectively return ciphertexts that would match the search. The server would still have to prove to the client that no other matching ciphertexts exist. Because Chefs extends the SFS read-only file system, it inherits the property that the client can verify when it has received all intended content (i.e., the whole truth) from the server. Therefore, the Chefs client downloads all the encrypted content and keys to perform the search itself.

**Sous-Chefs.** To determine the cost of key regression, we compare Chefs to a version of Chefs with key regression disabled. We call this strawman file system Sous-Chefs. Chefs and Sous-Chefs differ only in how they fetch group keys from the publisher. When using KR-SHA1 for key regression, Chefs fetches a 20-byte member state, encrypted in the client's public 1,024-bit RSA key with low exponent $e = 3$. Chefs then uses key regression to unwind and derive all past versions of the group key. In Sous-Chefs, we fetch at once all the derived group keys (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. We allow a Sous-Chefs client to request a single bulk transfer of every version of a group key to fairly amortize the cost of the transfer.

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

Each test asynchronously plays back 20 traces of a single client fetching the keys for the search workload. This effectively simulates the effect of 20 clients applying the same key distribution workload to the publisher. After all traces have completed, we record the effective number of

[5] `http://www.salon.com/press/release/`

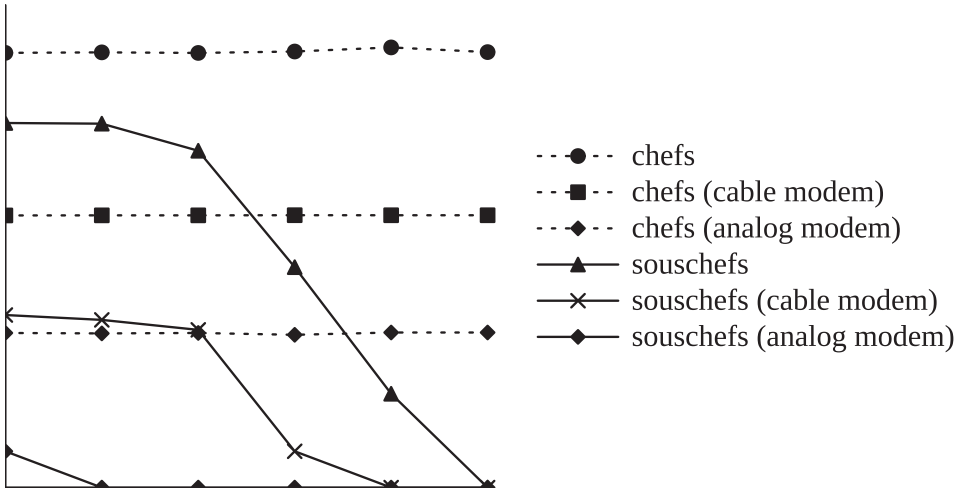

Figure 4: Aggregate publisher throughput for key distribution. Key regression enables a publisher to support many client-sessions per second. Chefs always performs better Sous-Chefs because key regression performance is effectively independent of the rate of membership turnover.

trace playbacks per second. The Sous-Chefs traces of fetching 10, $10^2$, $10^3$, $10^4$, $10^5$, and $10^6$ keys generate 4, 4, 5, 24, 200, and 1,966 asynchronous remote procedure calls from the client to the publisher respectively. Chefs always generates a single remote procedure call, regardless of the number of key versions.

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

Figure 5 shows that Chefs equals or outperforms Sous-Chefs for our search workload under several network conditions. In Sous-Chefs, the network transfer time dominates client latency because of the sheer volume of keys transferred from the publisher to the client. There is no measurement for Sous-Chefs downloading 1,000,000 keys on an analog modem because the operation expectedly times out.

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%.

## 7 Conclusions

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

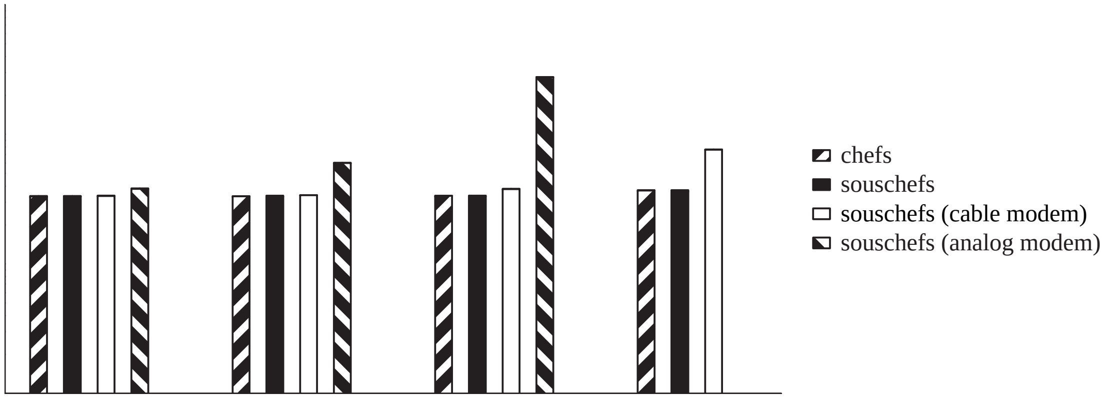

Figure 5: Single client latency to read 10,000 8 KB encrypted files and the associated content keys. Key regression maintains the same client latency as a system not using key regression. Under low-bandwidth, high-latency conditions, key regression can improve client latency linear with respect to the number of keys.

## Acknowledgments

We thank Ron Rivest and 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; and Frans Kaashoek for his guidance and unending support.

## References

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

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

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

[4] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In Y. G. Desmedt, editor, *Proceedings CRYPTO 94*, pages 341–358. Springer, 1994. Lecture Notes in Computer Science No. 839.

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

[6] M. Bellare and P. Rogaway. The game-playing technique. Cryptology ePrint Archive http://eprint.iacr.org/: Report 2004/331, 2004.

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

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

[9] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. 2005. See http://eprint.iacr.org/2005/018.

[10] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[11] A. Desai, A. Hevia, and Y. Yin. A practice-oriented treatment of pseudorandom number generators. In *Advances in Cryptology - EUROCRYPT 2002,*, volume 2332 of *Lecture Notes in Computer Science*, pages 368–383. Springer, 2002.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[27] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In J. Kilian, editor, *Advances in Cryptology – CRYPTO*, volume 2139, pages 41–62, Aug. 19–23, 2001.

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

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

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

[31] V. Shoup. Sequences of games: A tool for taming complexity in security proofs. Cryptology ePrint Archive `http://eprint.iacr.org/`: Report 2004/332, 2004.

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

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

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

# A Generalization of KR-SHA1 and proof of Theorem 5.2

Construction A.1 below shows a generalization of KR-SHA1 in which $\mathrm{SHA1}(\cdot)$ and $\mathrm{SHA1}(0^8\|\cdot)$ are respectively replaced by two random oracles, $H_1$ and $H_2$. For Construction A.1, in order for

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

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

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

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

Figure 6: Hash chains- and skeys-based algorithms for Construction A.1. $H_1$ and $H_2$ are random oracles. The $\mathsf{setup}$ algorithm uses the $\mathsf{unwind}$ algorithm defined in the second column.

$\mathsf{setup}$ and $\mathsf{wind}$ to be "efficient," we assume that $\mathsf{MW}$ has some "reasonable" value like $2^{20}$; in the asymptotic setting we would require that $\mathsf{MW}$ be polynomial in some security parameter. Besides KR-SHA1, one can envision a number of other natural instantiations of Construction A.1.

**Construction A.1** Let $H_1$: $\{0,1\}^k \rightarrow \{0,1\}^k$ and $H_2$: $\{0,1\}^k \rightarrow \{0,1\}^l$ be random oracles. Figure 6 shows how to construct a key regression scheme $\mathcal{KR} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ from $H_1$ and $H_2$; $\mathsf{MW}$ is a positive integer and a parameter of the construction. The derived key space for $\mathcal{KR}$ is $\mathsf{DK} = \{0,1\}^l$. ▌

The following theorem states that Construction A.1 is secure in the random oracle model for adversaries that make a reasonable number of queries to their random oracles.

**Theorem A.2** *The key regression scheme in Construction A.1 is secure in the random oracle model. Formally, let $H_1$: $\{0,1\}^k \rightarrow \{0,1\}^k$ and $H_2$: $\{0,1\}^k \rightarrow \{0,1\}^l$ be random oracles and let $\mathcal{KR}$ be the key regression scheme built from $H_1, H_2$ via Construction A.1. Then for any adversary $\mathcal{A}$ we have that*

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

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

**Proof of Theorem A.2:** Consider the experiments $\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}}$ and $\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}}$. Let $\mathsf{stm}_1, \mathsf{stm}_2, \ldots,$ $\mathsf{stm}_{\mathsf{MW}}$ denote the member states as computed by $\mathsf{setup}$, and let $w'$ denote the variable number of $\mathsf{WindO}$ oracle queries that $\mathcal{A}$ made in its $\mathsf{member}$ stage. Let $\mathcal{E}_1$ be the event in $\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}}$ that $w' \leq \mathsf{MW} - 1$ and that $\mathcal{A}$ queries either its $H_1$ or $H_2$ random oracles with some string $x \in \{\mathsf{stm}_{w'+1}, \ldots, \mathsf{stm}_{\mathsf{MW}}\}$. Let $\mathcal{E}_0$ be the event in $\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}}$ that $w' \leq \mathsf{MW} - 1$ and that $\mathcal{A}$ queries either its $H_1$ or $H_2$ random oracles with some string $x \in \{\mathsf{stm}_{w'+1}, \ldots, \mathsf{stm}_{\mathsf{MW}}\}$. Let $\mathcal{F}_1$ be the event in $\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}}$ that there exist two distinct indices $i, j \in \{1, \ldots, \mathsf{MW}\}$ such that $\mathsf{stm}_i = \mathsf{stm}_j$ and let $\mathcal{F}_0$ be the event in $\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}}$ that there exist two distinct indices $i, j \in \{1, \ldots, \mathsf{MW}\}$ such that $\mathsf{stm}_i = \mathsf{stm}_j$.

We claim that

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} \;\leq\; \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{F}_1\right] + \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1 \wedge \overline{\mathcal{F}_1}\right] \;, \qquad (1)$$

that

$$\Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{F}_1\right] \;\leq\; \frac{(\mathsf{MW})^2}{2^{k+1}} \;, \qquad (2)$$

and that

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1 \wedge \overline{\mathcal{F}_1}\right] \;\le\; \frac{q \cdot \mathsf{MW}}{2^k - \mathsf{MW} - q}\ , \tag{3}$$

from which the inequality in the theorem statement follows.

To justify Equation (1), let $\Pr_1[\,\cdot\,]$ and $\Pr_0[\,\cdot\,]$ denote the probabilities over $\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}}$ and $\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}}$, respectively. From Definition 4.1, we have

$$\begin{aligned}
\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{A}} &= \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1\right] - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1\right] \\
&= \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{F}_1\right] + \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1 \wedge \overline{\mathcal{F}_1}\right] \\
&\quad + \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_1} \wedge \overline{\mathcal{F}_1}\right] - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{F}_0\right] \\
&\quad - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_0 \wedge \overline{\mathcal{F}_0}\right] - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_0} \wedge \overline{\mathcal{F}_0}\right] \\
&\le \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{F}_1\right] + \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1 \wedge \overline{\mathcal{F}_1}\right] \\
&\quad + \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_1} \wedge \overline{\mathcal{F}_1}\right] - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_0} \wedge \overline{\mathcal{F}_0}\right]\ . \qquad (4)
\end{aligned}$$

By conditioning,

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_1} \wedge \overline{\mathcal{F}_1}\right] = \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \mid \overline{\mathcal{E}_1} \wedge \overline{\mathcal{F}_1}\right] \cdot \Pr_1\left[\overline{\mathcal{E}_1} \wedge \overline{\mathcal{F}_1}\right]$$

and

$$\Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_0} \wedge \overline{\mathcal{F}_0}\right] = \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \mid \overline{\mathcal{E}_0} \wedge \overline{\mathcal{F}_0}\right] \cdot \Pr_0\left[\overline{\mathcal{E}_0} \wedge \overline{\mathcal{F}_0}\right]\ .$$

Prior to the adversary causing the events $\mathcal{E}_1 \vee \mathcal{F}_1$ and $\mathcal{E}_0 \vee \mathcal{F}_0$ to occur in their respective experiments, $\mathcal{A}$'s view is identical in both experiments, meaning that

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

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

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \mid \overline{\mathcal{E}_1} \wedge \overline{\mathcal{F}_1}\right] \;=\; \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \mid \overline{\mathcal{E}_0} \wedge \overline{\mathcal{F}_0}\right]\ .$$

Consequently,

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_1} \wedge \overline{\mathcal{F}_1}\right] \;=\; \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_0} \wedge \overline{\mathcal{F}_0}\right]\ .$$

Combining the above equation with Equation (4) gives Equation (1).

Returning to Equation (2), we first note that

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{F}_1\right] \;\le\; \Pr_1\left[\mathcal{F}_1\right]\ .$$

If we consider the event $\mathcal{F}_1$, we note that the $\mathsf{setup}$ algorithm selects the points $\mathsf{stm}_{\mathsf{MW}}$, $\mathsf{stm}_{\mathsf{MW}-1}$, $\mathsf{stm}_{\mathsf{MW}-2}$, and so on, uniformly at random from $\{0,1\}^k$ until a collision occurs. Since this is exactly the standard birthday paradox [4], we can upper bound $\Pr_1[\mathcal{F}_1]$ as

$$\Pr_1\left[\mathcal{F}_1\right] \;\le\; \frac{(\mathsf{MW})^2}{2^{k+1}}\ .$$

```
Algorithm unwind(stm)
  $x \leftarrow G(\mathsf{stm})$
  $\mathsf{stm}' \leftarrow$ first $k$ bits of $x$
  Return $\mathsf{stm}'$
```

```
Algorithm keyder(stm)
  $x \leftarrow G(\mathsf{stm})$
  $K \leftarrow$ last $l$ bits of $x$
  Return $K$
```

Figure 7: The unwind and keyder algorithms for Construction B.1. $G$: $\{0,1\}^k \rightarrow \{0,1\}^{k+l}$ is a function. The setup and wind algorithms are as in Figure 6 except that setup and wind do not receive access to any random oracles.

Equation (2) follows.

To justify Equation (3), we begin by noting that

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1 \wedge \overline{\mathcal{F}_1}\right] \;\leq\; \Pr_1\left[\,\mathcal{E}_1 \mid \overline{\mathcal{F}_1}\,\right] \cdot \Pr_1\left[\overline{\mathcal{F}_1}\right] \;\leq\; \Pr_1\left[\,\mathcal{E}_1 \mid \overline{\mathcal{F}_1}\,\right] \;.$$

Consider the adversary $\mathcal{A}$ in $\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}}$ and assume that $\mathcal{F}_1$ does not occur. Consider any snapshot of the entire state of $\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}}$ before $\mathcal{A}$ causes $\mathcal{E}_1$ to occur, and let $q'$ denote the number of $H_1$ and $H_2$ oracle queries that $\mathcal{A}$ has made prior to the snapshot being taken. Then the member states $\mathsf{stm}_{w'+1}, \ldots, \mathsf{stm}_{\mathsf{MW}}$ are restricted only in that they are distinct strings from $\{0,1\}^k$ and that none of the strings are from $\{\mathsf{stm}_1, \ldots, \mathsf{stm}_{w'}\}$ or the set of $\mathcal{A}$'s $q'$ queries to its random oracles; i.e., the member states that $\mathcal{A}$ 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 $x \in \{\mathsf{stm}_{w'+1}, \ldots, \mathsf{stm}_{\mathsf{MW}}\}$, i.e., the probability that $\mathcal{A}$'s oracle query would cause $\mathcal{E}_1$ to occur, is at most $(\mathsf{MW} - w')/(2^k - (w' + q')) \leq \mathsf{MW}/(2^k - \mathsf{MW} - q')$. Summing over all of $\mathcal{A}$'s $q$ random oracle queries and taking an upper bound, we have

$$\Pr_1\left[\,\mathcal{E}_1 \mid \overline{\mathcal{F}_1}\,\right] \;\leq\; \frac{q \cdot \mathsf{MW}}{2^k - \mathsf{MW} - q} \;,$$

which completes the proof. ▌

**Proof of Theorem 5.2:** Theorem 5.2 follows immediately from Theorem A.2 since the latter makes a more general statement. ▌

## B Generalization of KR-AES and proof of Theorem 5.4

Construction B.1 below generalizes KR-AES, and is essentially one of Bellare and Yee's [7] forward secure PRGs in reverse. Construction B.1 uses a pseudorandom bit generator, which 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, $k, l \geq 1$. Pseudorandom bit generators were defined first in [8] and lifted to the concrete setting in [11]. As with Construction A.1, 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. To instantiate KR-AES from Construction B.1, we set $k = l = 128$ and, for any $X \in \{0,1\}^{128}$, we define $G$ as $G(X) = \mathrm{AES}_X(0^{128}) \| \mathrm{AES}_X(1^{128})$. Numerous other instantiations exist. The security proof for Construction B.1 is in the standard, as opposed to the random oracle, model.

**Construction B.1** Let $G$: $\{0,1\}^k \to \{0,1\}^{k+l}$ be a pseudorandom bit generator. Figure 7 shows how to construct a key regression scheme $\mathcal{KR} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ from $G$; $\mathsf{MW}$ is a positive integer and a parameter of the construction. The derived key space for the scheme $\mathcal{KR}$ is $\mathsf{DK} = \{0,1\}^l$. ▌

Toward proving the security of Construction B.1, we begin by defining our security assumptions on the base PRG [8, 34, 7]. If $\mathcal{A}$ is an adversary, we let

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

denote the *prg-advantage* of $\mathcal{A}$ in attacking $G$. Under the concrete security approach [4], there is no formal definition of what it means for $G$ to be a "secure PRG," but in discussions this phrase should be taken to mean that, for any $\mathcal{A}$ attacking $G$ with resources (running time, size of code) limited to "practical" amounts, the prg-advantage of $\mathcal{A}$ is "small." Formal results are stated with concrete bounds.

**Theorem B.2** *If $G$: $\{0,1\}^k \to \{0,1\}^{k+l}$ is a secure PRG, then the key regression scheme $\mathcal{KR}$ built from G via Construction B.1 is KR-secure. Concretely, given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$, we can construct an adversary $\mathcal{B}$ attacking G such that*

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

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

For our proof of Theorem B.2, we remark that the internal structure of the member states and derived keys in Construction B.1 is very similar to the internal structure of the states and output bits in a forward-secure pseudorandom bit generator, as defined in [7] and recalled below. Our proof therefore proceeds first by showing how to build a secure key regression scheme from any forward-secure pseudorandom bit generator, essentially by running the forward-secure pseudorandom bit generator in reverse during the key regression scheme's $\mathsf{setup}$ algorithm (Construction B.3). This intermediate result suggests that future work in forward-secure pseudorandom bit generators could have useful applications to key regression schemes. To prove Theorem B.2, we then combine this intermediate result with a lemma in [7] that shows how to create a forward-secure pseudorandom bit generator from a conventional pseudorandom bit generator.

Before proving Theorem B.2, we first use Theorem B.2 to prove Theorem 5.4.

**Proof of Theorem 5.4:** To instantiate $\mathsf{KR\text{-}AES}$ from Construction B.1, we set $k = l = 128$ and, for any $X \in \{0,1\}^{128}$, we define $G$ as $G(X) = \mathrm{AES}_X(0^{128}) \| \mathrm{AES}_X(1^{128})$.

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

$$\mathbf{Adv}^{\mathrm{prg}}_{G,\mathcal{B}} \;\leq\; \mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES},\mathcal{C}} + 2^{-128} \tag{5}$$

and $\mathcal{C}$ makes two oracle queries and uses within a small constant factor of the resources of $\mathcal{B}$. Theorem 5.4 follows from this claim and Theorem B.2.

We now justify our claim above. Let $\mathrm{Func}(l,l)$ 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 $f$: $\{0,1\}^{128} \to \{0,1\}^{128}$ is $\mathcal{C}$'s oracle. Adversary $\mathcal{C}$ then returns the same bit that $\mathcal{B}$ returns.

Note that

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

since, when $\mathcal{C}$'s oracle is $\mathrm{AES}_K(\cdot)$, $\mathcal{C}$ runs $\mathcal{B}$ with input $\mathrm{AES}_K(0^{128}) \| \mathrm{AES}_K(1^{128})$, for a randomly selected key $K$, which has the same distribution as $G(K)$ for a randomly selected key $K$. Additionally,

$$\Pr\left[ x \xleftarrow{\$} \{0,1\}^{256} \;:\; \mathcal{B}(x) = 1 \right] \;=\; \Pr\left[ g \xleftarrow{\$} \mathrm{Func}(128,128) \;:\; \mathcal{C}^{g(\cdot)} = 1 \right]$$

since, when $\mathcal{C}$'s oracle is a random function from $\{0,1\}^{128}$ to $\{0,1\}^{128}$, it runs $\mathcal{B}$ with a random 256-bit string.

Expanding the definition of $\mathbf{Adv}^{\mathrm{prg}}_{G,\mathcal{B}}$ and substituting the above equalities, we have

$$\begin{aligned} \mathbf{Adv}^{\mathrm{prg}}_{G,\mathcal{B}} &= \Pr\left[ K \xleftarrow{\$} \{0,1\}^{128} \,;\, x \leftarrow G(K) \;:\; \mathcal{B}(x) = 1 \right] - \Pr\left[ x \xleftarrow{\$} \{0,1\}^{256} \;:\; \mathcal{B}(x) = 1 \right] \\ &= \Pr\left[ K \xleftarrow{\$} \{0,1\}^{128} \;:\; \mathcal{C}^{\mathrm{AES}_K(\cdot)} = 1 \right] - \Pr\left[ g \xleftarrow{\$} \mathrm{Func}(128,128) \;:\; \mathcal{C}^{g(\cdot)} = 1 \right] . \end{aligned}$$

If we subtract and add $\Pr\left[ g \xleftarrow{\$} \mathrm{Perm}(128) \;:\; \mathcal{C}^{g(\cdot)} = 1 \right]$ and apply the definition of $\mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES},\mathcal{C}}$, we get

$$\begin{aligned} \mathbf{Adv}^{\mathrm{prg}}_{G,\mathcal{B}} &= \Pr\left[ K \xleftarrow{\$} \{0,1\}^{128} \;:\; \mathcal{C}^{\mathrm{AES}_K(\cdot)} = 1 \right] - \Pr\left[ g \xleftarrow{\$} \mathrm{Perm}(128) \;:\; \mathcal{C}^{g(\cdot)} = 1 \right] \\ &\quad + \Pr\left[ g \xleftarrow{\$} \mathrm{Perm}(128) \;:\; \mathcal{C}^{g(\cdot)} = 1 \right] - \Pr\left[ g \xleftarrow{\$} \mathrm{Func}(128,128) \;:\; \mathcal{C}^{g(\cdot)} = 1 \right] \\ &= \mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES},\mathcal{C}} \\ &\quad + \Pr\left[ g \xleftarrow{\$} \mathrm{Perm}(128) \;:\; \mathcal{C}^{g(\cdot)} = 1 \right] - \Pr\left[ g \xleftarrow{\$} \mathrm{Func}(128,128) \;:\; \mathcal{C}^{g(\cdot)} = 1 \right] . \end{aligned}$$

Using the standard PRF/PRP switching result from [4], re-proven in [6, 31], and the fact that $\mathcal{C}$ makes only two oracle queries, the above simplifies to Equation (5), completing the proof. ▌

## B.1 Forward-secure pseudorandom generators

In [7], Bellare and Yee define stateful pseudorandom bit generators and describe what it means for a stateful pseudorandom bit generator to be forward-secure. Intuitively a stateful PRG is forward-secure if even adversaries that are given the generator's current state cannot distinguish previous outputs from random.

**Syntax.** A stateful PRG consists of two algorithms: $\mathcal{SBG} = (\mathsf{seed}, \mathsf{next})$. The randomized setup algorithm returns an initial state; we write this as $\mathsf{stg} \xleftarrow{\$} \mathsf{seed}$. The deterministic next step algorithm takes a state as input and returns a new state and an output from $\mathsf{OutSp}_{\mathcal{SBG}}$, or the pair $(\bot, \bot)$; we write this as $(\mathsf{stg}', K) \leftarrow \mathsf{next}(\mathsf{stg})$. We require that the set $\mathsf{OutSp}_{\mathcal{SBG}}$ is efficiently samplable. $\mathsf{MaxLen}_{\mathcal{SBG}} \in \{1, 2, \ldots\} \cup \{\infty\}$ denotes the maximum number of output blocks that $\mathcal{SBG}$ is designed to produce.

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

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

```
Algorithm setup
    $\mathsf{stg}_{\mathsf{MW}} \xleftarrow{\$} \mathsf{seed}$
    For $i = \mathsf{MW}$ downto 2 do
        $(\mathsf{stg}_{i-1}, K_{i-1}) \leftarrow \mathsf{next}(\mathsf{stg}_i)$
    $\mathsf{stp} \leftarrow \langle 1, \mathsf{stg}_1, \ldots, \mathsf{stg}_{\mathsf{MW}} \rangle$
    Return $\mathsf{stp}$
```

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

```
Algorithm unwind(stm)
    $(\mathsf{stm}', K) \leftarrow \mathsf{next}(\mathsf{stm})$
    Return $\mathsf{stm}'$

Algorithm keyder(stm)
    $(\mathsf{stm}', K) \leftarrow \mathsf{next}(\mathsf{stm})$
    Return $K$
```

Figure 8: Algorithms for KR-SBG (Construction B.3). Construction KR-SBG demonstrates how to build a KR-secure key regression scheme from any FSPRG-secure stateful bit generator $\mathcal{SBG} = (\mathsf{seed}, \mathsf{next})$. All algorithms may have access to the next algorithm. The setup algorithm assumes access to the seed algorithms, and that the unwind and keyder algorithms do not have access to seed.

---

```
Experiment $\mathbf{Exp}^{\text{fsprg-}b}_{\mathcal{SBG},\mathcal{A}}$
    $\mathsf{stg} \xleftarrow{\$} \mathsf{seed}$
    $\mathsf{st} \xleftarrow{\$} \mathcal{A}^{\mathsf{NextO}_b}(\mathsf{find})$
    $b' \xleftarrow{\$} \mathcal{A}(\mathsf{guess}, \mathsf{stg}, \mathsf{st})$
    Return $b'$
```

```
Oracle $\mathsf{NextO}_b$
    $(\mathsf{stg}, K) \leftarrow \mathsf{next}(\mathsf{stg})$
    If $b = 0$ then $K \xleftarrow{\$} \mathsf{OutSp}_{\mathcal{SBG}}$
    Return $K$
```

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

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

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

## B.2 Key regression from forward-secure pseudorandom bit generators

**Construction B.3** [KR-SBG] Given a stateful generator $\mathcal{SBG} = (\mathsf{seed}, \mathsf{next})$, we can construct a key regression scheme $\mathsf{KR\text{-}SBG} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$ as follows. For the construction, we set MW to a positive integer at most $\mathsf{MaxLen}_{\mathcal{SBG}}$; MW is a parameter of our construction. The derived key space for KR-SBG is $\mathsf{DK} = \mathsf{OutSp}_{\mathcal{SBG}}$. The algorithms for $\mathcal{KR}$ are shown in Figure 8. ▮

**Lemma B.4** If $\mathcal{SBG}$ is FSPRG-secure, then $\mathcal{KR}$ built from $\mathcal{SBG}$ via KR-SBG (Construction B.3) is KR-secure. Concretely, given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{SBG}$ such that

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

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

**Proof of Lemma B.4:** The adversary $\mathcal{B}$ is shown in Figure 9. The main idea is that if $\mathcal{B}$ correctly guesses the number of WindO queries that $\mathcal{A}$ will make, then $\mathcal{B}$'s simulation is perfect for either

```
Adversary $\mathcal{B}^{\mathsf{NextO}_b}(\mathsf{find})$
    $q' \xleftarrow{\$} \{0, 1, \ldots, q\}$
    For $i = \mathsf{MW}$ downto $q' + 1$ do
        $K_{i-1} \leftarrow \mathsf{NextO}_b$
    Return $\langle q', K_{q'}, \ldots, K_{\mathsf{MW}-1} \rangle$

Adversary $\mathcal{B}(\mathsf{guess}, \mathsf{stg}, \mathsf{st})$
    Parse $\mathsf{st}$ as $\langle q', K_{q'}, \ldots, K_{\mathsf{MW}-1} \rangle$
    $\mathsf{stg}_{q'} \leftarrow \mathsf{stg}$
    For $i = q'$ downto 2 do
        $(\mathsf{stg}_{i-1}, K_{i-1}) \leftarrow \mathsf{next}(\mathsf{stg}_i)$
    $i \leftarrow 0$
    $\mathsf{bad} \leftarrow \mathsf{false}$
    $\mathsf{st}_{\mathcal{A}} \xleftarrow{\$} \mathcal{A}^{\mathsf{SimWindO}}(\mathsf{member})$
    If $i \neq q'$ then $\mathsf{bad} \leftarrow \mathsf{true}$
    If $\mathsf{bad} = \mathsf{true}$ then return 0
    $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{SimKeyderO}}(\mathsf{non\text{-}member}, \mathsf{st}_{\mathcal{A}})$
    Return $b'$

Oracle SimWindO
    If $i \geq q'$ then $\mathsf{bad} \leftarrow \mathsf{true}$
    If $i \geq \mathsf{MW}$ or $\mathsf{bad} = \mathsf{true}$
        then return $\perp$
    Else $i \leftarrow i + 1$
    return $\mathsf{stg}_i$

Oracle SimKeyderO
    If $i \geq \mathsf{MW}$ then return $\perp$
    $i \leftarrow i + 1$
    Return $K_{i-1}$
```

Figure 9: The adversary $\mathcal{B}$ in the proof of Theorem B.4.

choice of bit $b$. If $\mathcal{B}$ does not correctly guess the bit 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 $\mathcal{A}$ makes since wind is defined to return $(\perp, \perp)$ after MW invocations.

Formally, we claim that

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1\right] = (q+1) \cdot \Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}} = 1\right] \tag{6}$$

$$\Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1\right] = (q+1) \cdot \Pr\left[\mathbf{Exp}^{\text{fsprg-0}}_{\mathcal{SBG},\mathcal{A}} = 1\right], \tag{7}$$

from which it follows that

$$\begin{aligned}
\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{A}} &= \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1\right] - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1\right] \\
&= (q+1) \cdot \left(\Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}} = 1\right] - \Pr\left[\mathbf{Exp}^{\text{fsprg-0}}_{\mathcal{SBG},\mathcal{A}} = 1\right]\right) \\
&\leq (q+1) \cdot \mathbf{Adv}^{\text{fsprg}}_{\mathcal{SBG},\mathcal{B}}
\end{aligned}$$

as desired.

It remains to justify Equation (6), Equation (7), and the resources of $\mathcal{B}$. Let $\mathcal{E}_1$ and $\mathcal{E}_0$ respectively denote the events that $\mathcal{B}$ sets bad to true in the experiments $\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}}$ and $\mathbf{Exp}^{\text{fsprg-0}}_{\mathcal{SBG},\mathcal{A}}$, i.e., when $\mathcal{B}$ fails to correctly guess the number of wind oracle queries that $\mathcal{A}$ makes. Let $\Pr_1[\cdot]$ and $\Pr_0[\cdot]$

```
Algorithm seed
    $\mathsf{stg}_0 \xleftarrow{\$} \{0,1\}^k$
    return $\mathsf{stg}_0$

Algorithm next($\mathsf{stg}_i$)
    $r \xleftarrow{\$} G(\mathsf{stg}_i)$
    $\mathsf{stg}_{i+1} \leftarrow$ first $k$ bits of $r$
    $K \leftarrow$ last $l$ bits of $r$
    return $(\mathsf{stg}_{i+1}, K)$
```

Figure 10: Algorithms for Construction B.5.

respectively denote probabilities over $\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}}$ and $\mathbf{Exp}^{\text{fsprg-0}}_{\mathcal{SBG},\mathcal{A}}$. We now claim that

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1\right] = \Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}} = 1 \mid \overline{\mathcal{E}_1}\right] \tag{8}$$

$$= \Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}} = 1 \wedge \overline{\mathcal{E}_1}\right] \cdot \frac{1}{\Pr_1\left[\overline{\mathcal{E}_1}\right]} \tag{9}$$

$$= (q+1) \cdot \Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}} = 1 \wedge \overline{\mathcal{E}_1}\right] \tag{10}$$

$$= (q+1) \cdot \left(\Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}} = 1 \wedge \overline{\mathcal{E}_1}\right] + \Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}} = 1 \wedge \mathcal{E}_1\right]\right) \tag{11}$$

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

Equation (8) 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 $\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}}$ runs $\mathcal{A}$ exactly as $\mathcal{A}$ would be run in $\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}}$. Equation (9) follows from conditioning off $\Pr_1\left[\overline{\mathcal{E}_1}\right]$ and Equation (10) is true because $\mathcal{B}$ chooses $q'$ from $q+1$ possible values and therefore $\Pr_1\left[\overline{\mathcal{E}_1}\right] = 1/(q+1)$. To justify Equation (11), note that $\Pr\left[\mathbf{Exp}^{\text{fsprg-1}}_{\mathcal{SBG},\mathcal{B}} = 1 \wedge \mathcal{E}_1\right] = 0$ 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 (6).

To justify Equation (7), note that

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

since when the event $\mathcal{E}_0$ does not occur, $\mathcal{B}$ in $\mathbf{Exp}^{\text{fsprg-0}}_{\mathcal{SBG},\mathcal{B}}$ runs $\mathcal{A}$ exactly as $\mathcal{A}$ would be run in $\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}}$. The remaining justification for Equation (7) is analogous to our justification of Equation (6) 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 $\mathsf{setup}$ algorithm itself, which involves querying its oracle up to $\mathsf{MW}$ times. ▮

## B.3 Forward-secure pseudorandom bit generators from standard PRGs

**Construction B.5 [Construction 2.2 of [7].]** Given a PRG $G : \{0,1\}^k \to \{0,1\}^{k+l}$ we can construct a FSPRG $\mathcal{SBG} = (\mathsf{seed}, \mathsf{next})$ as described in Figure 10. The output space of $\mathcal{SBG}$ is $\mathsf{OutSp}_{\mathcal{SBG}} = \{0,1\}^l$ and $\mathsf{MaxLen}_{\mathcal{SBG}} = \infty$. ▮

The following theorem comes from Bellare and Yee [7] except that we treat $q$ as a parameter of the adversary and we allow the trivial case that $q = 0$.

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

Alg. wind^H(stp)
    Parse stp as $\langle N, e, d, S \rangle$
    $S' \leftarrow S^d \bmod N$
    $\mathsf{stp}' \leftarrow \langle N, e, d, S' \rangle$
    $\mathsf{stm} \leftarrow \langle N, e, S \rangle$
    Return $(\mathsf{stp}', \mathsf{stm})$
```

```
Alg. unwind^H(stm)
    Parse stm as $\langle N, e, S \rangle$
    $S' \leftarrow S^e \bmod N$
    $\mathsf{stm}' \leftarrow \langle N, e, S' \rangle$
    Return stm'

Alg. keyder^H(stm)
    Parse stm as $\langle N, e, S \rangle$
    $K \leftarrow H(S)$
    Return $K$
```

Figure 11: Algorithms for Construction C.1. $H$ is a random oracle.

**Lemma B.6 [Theorem 2.3 of [7].]** Let $G : \{0,1\}^k \rightarrow \{0,1\}^{k+l}$ be a PRG, and let $\mathcal{SBG}$ be the FSPRG built using $G$ according to Construction B.5. Given an adversary $\mathcal{A}$ attacking $\mathcal{SBG}$ that makes at most $q$ queries to its oracle, we can construct an adversary $\mathcal{B}$ such that

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

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

### B.4 Proof of Theorem B.2

**Proof:** Construction B.1 is exactly Construction B.3 built from the forward secure pseudorandom bit generator defined by Construction B.5. The theorem statement therefore follows from Lemma B.4 and Lemma B.6. ▮

## C Generalization of KR-RSA and proof of Theorem 5.6

While we consider our previous constructions to be practical in most cases, in some cases one might need properties that they do not achieve. For example, if MW is large, maintaining the owner states may require a non-trivial amount of space or time. Also, both the random oracle and PRG based constructions only handle a limited number of winds. We address these concerns by presenting a construction based on RSA that can handle an infinite number of winds and that requires a small amount of space to store its state.

**Construction C.1** Given an RSA key generator $\mathcal{K}_{\text{rsa}}$ for some security parameter $k$ and a random oracle $H\colon \mathbb{Z}_{2^k} \rightarrow \{0,1\}^l$, Figure 11 shows how to construct a key regression scheme $\mathcal{KR} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$. The derived key space for $\mathcal{KR}$ is $\mathsf{DK} = \{0,1\}^l$. ▮

**Theorem C.2** *If $\mathcal{K}_{rsa}$ is an RSA key generator with security parameter $k$, then $\mathcal{KR}$ built from $\mathcal{K}_{rsa}$ via Construction C.1 is KR-secure under the RSA assumption. Concretely, given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{K}_{rsa}$ such that*

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

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

The proof of Theorem C.2 uses the two following Lemmas.

**Lemma C.3** If a key regression scheme is secure when an adversary is limited to one $\mathsf{KeyderO}$ oracle query, then the key regression scheme is secure when an adversary is allowed multiple $\mathsf{KeyderO}$ oracle queries. Concretely, let $\mathcal{KR}$ be a key regression scheme. Given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$ that makes at most $q_1$ queries to $\mathsf{WindO}$ and $q_2$ queries to $\mathsf{KeyderO}$, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{KR}$ such that

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} \;\leq\; q_2 \cdot \mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{B}} \;, \tag{12}$$

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

**Lemma C.4** If $\mathcal{K}_{\mathrm{rsa}}$ is an RSA key generator with security parameter $k$, then the key regression scheme $\mathcal{KR}$ built from $\mathcal{K}_{\mathrm{rsa}}$ via Construction C.1 is KR-secure assuming that $\mathcal{K}_{\mathrm{rsa}}$ is one-way. Concretely, given an adversary $\mathcal{A}$ attacking $\mathcal{KR}$ that makes at most one key derivation oracle query, we can construct an adversary $\mathcal{B}$ attacking $\mathcal{K}_{\mathrm{rsa}}$ such that

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} \leq (q+1) \cdot \mathbf{Adv}^{\mathrm{rsa\text{-}ow}}_{\mathcal{K}_{\mathrm{rsa}},\mathcal{B}} \;, \tag{13}$$

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

**Proof of Theorem C.2:** The proof of Theorem C.2 follows from Lemma C.3 and Lemma C.4. Note that for the application of Lemma C.3 we set $q_1 = q$ and $q_2 = q$, meaning the adversary $\mathcal{B}$ from Lemma C.3 may make up to $2q - 1$ queries to its $\mathsf{WindO}$ oracle, or $2q$ if $q = 0$. ▮

**Proof of Theorem 5.6:** Theorem 5.6 is Theorem C.2 with $l = 160$. ▮

## C.1 Proof of Lemma C.3

**Proof:** We consider the case where $q_2 = 0$ separately. If $q_2 = 0$ then

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

since the adversary $\mathcal{A}$'s view in the experiments $\mathbf{Exp}^{\mathrm{kr\text{-}1}}_{\mathcal{KR},\mathcal{A}}$ and $\mathbf{Exp}^{\mathrm{kr\text{-}0}}_{\mathcal{KR},\mathcal{A}}$ is identical. Therefore, when $q_2 = 0$,

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} &= \Pr\left[\mathbf{Exp}^{\mathrm{kr\text{-}1}}_{\mathcal{KR},\mathcal{A}} = 1\right] - \Pr\left[\mathbf{Exp}^{\mathrm{kr\text{-}0}}_{\mathcal{KR},\mathcal{A}} = 1\right] \\
&= 0 \\
&= q_2 \cdot \mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{B}}
\end{aligned}$$

for all adversaries $\mathcal{B}$.

```
$\textbf{Experiment } \mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i}$
    Pick random oracle $H$
    $i \leftarrow 0$
    $\mathsf{stp} \xleftarrow{\$} \mathsf{setup}^H$
    $\mathsf{st} \xleftarrow{\$} \mathcal{A}^{\mathsf{HWindO},H}(\mathsf{member})$
    $j \leftarrow 0$
    $b' \xleftarrow{\$}$
        $\mathcal{A}^{\mathsf{HKeyderO}_i,H}(\mathsf{non\text{-}member}, \mathsf{st})$
    Return $b'$
```

```
$\textbf{Oracle } \mathsf{HWindO}$
    $i \leftarrow i+1$
    If $i > \mathsf{MW}$ then
        return $\perp$
    $(\mathsf{stp}, \mathsf{stm}) \xleftarrow{\$} \mathsf{wind}^H(\mathsf{stp})$
    Return $\mathsf{stm}$
```

```
$\textbf{Oracle } \mathsf{HKeyderO}_i$
    $i \leftarrow i+1$
    If $i > \mathsf{MW}$ then
        return $\perp$
    $(\mathsf{stp}, \mathsf{stm}) \xleftarrow{\$} \mathsf{wind}^H(\mathsf{stp})$
    If $j < i$ then
        $K \leftarrow \mathsf{keyder}^H(\mathsf{stm})$
    Else
        $K \xleftarrow{\$} \mathsf{DK}$
    $j \leftarrow j+1$
    Return $K$
```

Figure 12: Hybrid experiments for the proof of Lemma C.3.

We now restrict our analysis to the case where $q_2 \geq 1$. Consider the experiments $\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i}$ in Figure 12, $i \in \{0, \ldots, q_2\}$. When $i = q_2$, $\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i}$ uses $\mathsf{keyder}$ to reply to all of $\mathcal{A}$'s $\mathsf{HKeyderO}$ oracle queries, which means that

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1\right] = \Pr\left[\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},q_2} = 1\right] .$$

On the other hand, when $i = 0$, $\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i}$ replies to all of $\mathcal{A}$'s $\mathsf{HKeyderO}$ oracle queries with random values from $\mathsf{DK}$, which means that

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

From these two equations we conclude that

$$\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{A}} = \Pr\left[\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},q_2} = 1\right] - \Pr\left[\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},0} = 1\right] . \tag{14}$$

Consider now the adversary $\mathcal{B}$ in Figure 13. We claim that

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{B}} = 1\right] = \frac{1}{q_2} \cdot \sum_{i=0}^{q_2-1} \Pr\left[\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i+1} = 1\right] \tag{15}$$

and

$$\Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{B}} = 1\right] = \frac{1}{q_2} \cdot \sum_{i=0}^{q_2-1} \Pr\left[\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i} = 1\right] . \tag{16}$$

Subtracting Equation (16) from Equation (15) and using Definition 4.1, we get

$$\begin{aligned}
\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{B}} &= \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{B}} = 1\right] - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{B}} = 1\right] \\
&= \frac{1}{q_2} \cdot \left(\Pr\left[\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},q_2} = 1\right] - \Pr\left[\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},0} = 1\right]\right) . \qquad (17)
\end{aligned}$$

Equation (12) follows from combining Equation (14) with Equation (17).

It remains to justify Equation (15), Equation (16), and the resources of $\mathcal{B}$. To justify Equation (15), note that in the experiment $\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{B}}$, when $\mathcal{B}$ picks some value for $i$, the view of $\mathcal{A}$ becomes

```
Adversary $\mathcal{B}^{\mathsf{WindO},H}(\mathsf{member})$
    $i \xleftarrow{\$} \{0,\ldots,q_2-1\}$
    $l \leftarrow 0$
    Run $\mathcal{A}^{\mathsf{WindO}',H'}(\mathsf{member})$,
    replying to $\mathcal{A}$'s oracle queries as follows:
        For each query to $\mathsf{WindO}'$ do
            $\mathsf{stm} \xleftarrow{\$} \mathsf{WindO}$
            $l \leftarrow l+1$
            If $l > \mathsf{MW}$ then $\mathsf{stm} \leftarrow \bot$
            Return $\mathsf{stm}$ to $\mathcal{A}$
        For each query $x$ to $H'$ do
            $y \leftarrow H(x)$
            Return $y$ to $\mathcal{A}$
    Until $\mathcal{A}$ halts outputting a state $\mathsf{st}'$
    For $j = 0$ to $i-1$ do
        $\mathsf{stm} \xleftarrow{\$} \mathsf{WindO}$
        $K_j \leftarrow \mathsf{keyder}^H(\mathsf{stm})$
    $\mathsf{st} \leftarrow (\mathsf{st}', i, l, K_0, \ldots, K_{i-1})$
    Return $\mathsf{st}$
```

```
Adversary $\mathcal{B}^{\mathsf{KeyderO}_b,H}(\mathsf{non\text{-}member}, \mathsf{st})$
    Parse $\mathsf{st}$ as $(\mathsf{st}', i, l, K_0, \ldots, K_{i-1})$
    $j \leftarrow 0$
    Run $\mathcal{A}^{\mathsf{KeyderO}',H'}(\mathsf{non\text{-}member}, \mathsf{st}')$,
    replying to $\mathcal{A}$'s oracle queries as follows:
        For each query to $\mathsf{KeyderO}'$ do
            If $j < i$ then $K \leftarrow K_j$
            Else if $j = i$ then $K \leftarrow \mathsf{KeyderO}_b$
            Else $K \xleftarrow{\$} \mathsf{DK}$
            $j \leftarrow j+1$ ; $l \leftarrow l+1$
            If $l > \mathsf{MW}$ then $K \xleftarrow{\$} \bot$
            Return $K$ to $\mathcal{A}$
        For each query $x$ to $H'$ do
            $y \leftarrow H(x)$
            Return $y$ to $\mathcal{A}$
    Until $\mathcal{A}$ halts outputting a bit $b$
    Return $b$
```

Figure 13: Adversary $\mathcal{B}$ for the proof of Lemma C.3. We describe in the body an alternate description with reduced resource requirements.

equivalent to $\mathcal{A}$'s view in $\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i+1}$; namely, $\mathcal{A}$'s first $i+1$ queries to its $\mathsf{KeyderO}$ oracle will be computed using $\mathsf{keyder}$, and the remaining $\mathsf{KeyderO}$ oracle queries will return random values from $\mathsf{DK}$. More formally, if $I$ denotes the random variable for the $\mathcal{B}$'s selection for the variable $i \in \{0,\ldots,q_2-1\}$, then

$$\Pr\left[\,\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{B}} = 1 \;\middle|\; I = i\,\right] = \Pr\left[\,\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i+1} = 1\,\right]$$

for each $i \in \{0,\ldots,q_2-1\}$. Letting $\Pr_1[\,\cdot\,]$ denote the probability over $\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{B}}$, we then derive Equation (15) by conditioning off the choice of $i$:

$$\begin{aligned}
\Pr\left[\,\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{B}} = 1\,\right] &= \sum_{i=0}^{q_2-1} \Pr\left[\,\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{B}} = 1 \;\middle|\; I = i\,\right] \cdot \Pr_1\left[\,I = i\,\right] \\
&= \frac{1}{q_2} \cdot \sum_{i=0}^{q_2-1} \Pr\left[\,\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{B}} = 1 \;\middle|\; I = i\,\right] \\
&= \frac{1}{q_2} \cdot \sum_{i=0}^{q_2-1} \Pr\left[\,\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i+1} = 1\,\right]
\end{aligned}$$

The justification for Equation (16) is similar. When $\mathcal{B}$ picks some value for $i$ in $\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{B}}$, the view of $\mathcal{A}$ in $\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{B}}$ becomes equivalent to $\mathcal{A}$'s view in $\mathbf{ExpH}_{\mathcal{KR},\mathcal{A},i}$ since in both cases the responses to $\mathcal{A}$'s first $i$ (not $i+1$ this time) queries to its $\mathsf{KeyderO}$ oracle will be computed using $\mathsf{keyder}$, and the remaining $\mathsf{KeyderO}$ oracle queries will return random values from $\mathsf{DK}$.

```
Adversary $\mathcal{B}(y, e, N)$
    $\mathsf{bad} \leftarrow \mathsf{false}$
    $\alpha \leftarrow \bot$
    $j \leftarrow 0$
    $w \xleftarrow{\$} \{0, 1, 2, \ldots, q\}$
    $\mathsf{stm}_w \leftarrow y$
    For $i = w - 1$ downto 1 do
        $\mathsf{stm}_i \leftarrow (\mathsf{stm}_{i+1})^e \mod N$
    $\mathsf{st} \xleftarrow{\$} \mathcal{A}^{\mathsf{SimWindO},\mathsf{SimH}}(\mathsf{member})$
    If $j \neq w$ then
        $\mathsf{bad} \leftarrow \mathsf{true}$
        Return $\bot$
    $b \xleftarrow{\$} \mathcal{A}^{\mathsf{SimKeyderO},\mathsf{SimH}}(\mathsf{non\text{-}member}, \mathsf{st})$
    Return $\alpha$
```

```
Oracle SimWindO
    $j \leftarrow j + 1$
    If $j \leq w$ then return $\mathsf{stm}_j$
    Else return $\bot$

Oracle SimKeyderO
    $K \xleftarrow{\$} \mathsf{DK}$
    Return $K$

Oracle SimH$(x)$
    If $x^e = y \mod n$ then $\alpha \leftarrow x$
    If $H[x]$ undefined then
        $H[x] \xleftarrow{\$} \mathsf{DK}$
    Return $H[x]$
```

Figure 14: The adversary $\mathcal{B}$ in the proof of Lemma C.4.

We now turn to the resource requirements of $\mathcal{B}$. The pseudocode for $\mathcal{B}$ in Figure 13 suggests that $\mathcal{B}$ might invoke WindO and keyder up to $q_2$ times more than $\mathcal{A}$ (since the last for loop of $\mathcal{B}$'s member stage runs for up to $q_2$ interactions even though $\mathcal{A}$ may not make that many KeyderO oracle queries). We describe $\mathcal{B}$ this way since we feel that Figure 13 better captures the main idea behind our proof and what $\mathcal{B}$ does. Equivalently, $\mathcal{B}$ could split $\mathcal{A}$'s non-member stage between its ($\mathcal{B}$'s) own member and non-member stages and invoke WindO and keyder only the number of times that it needs to simulate $i$ of $\mathcal{A}$'s $\mathsf{KeyderO}_1$ oracle queries. When viewed this way, $\mathcal{B}$ uses resources equivalent, within a constant factor, to the resources of $\mathcal{A}$. ▌

## C.2 Proof of Lemma C.4

**Proof:** Consider the experiments $\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}}$ and $\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}}$; let $(N, e, S_1), (N, e, S_2), \ldots, (N, e, S_{w'})$ denote the responses to $\mathcal{A}$'s wind oracle queries when $\mathcal{A}$ is in the member stage, $w' \in \{0, 1, \ldots, q\}$. Let $\mathcal{E}_1$ and $\mathcal{E}_0$ respectively be the events in $\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}}$ and $\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}}$ that $\mathcal{A}$ queries its random oracle with a value $S$ such that $S^e \equiv S_{w'} \bmod N$. We claim that

$$\mathbf{Adv}^{\text{kr}}_{\mathcal{KR},\mathcal{A}} \;=\; \Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1\right] - \Pr\left[\mathbf{Exp}^{\text{kr-0}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_0\right] . \tag{18}$$

Consider now the adversary $\mathcal{B}$ in Figure 14. We additionally claim that

$$\Pr\left[\mathbf{Exp}^{\text{kr-1}}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1\right] \;\leq\; (q+1) \cdot \Pr\left[\mathbf{Exp}^{\text{rsa-ow}}_{\mathcal{K}_{\text{rsa}},\mathcal{B}} = 1\right] . \tag{19}$$

Combining these two equations and the definition of security for $\mathcal{K}_{\text{rsa}}$ gives Equation (13).

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

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{KR},\mathcal{A}} &= \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1\right] - \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}} = 1\right] \\
&= \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_1}\right] + \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1\right] \\
&\quad - \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_0}\right] - \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_0\right] . \qquad (20)
\end{aligned}$$

By conditioning,

$$\Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_1}\right] = \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \mid \overline{\mathcal{E}_1}\right] \cdot \Pr\nolimits_1\left[\overline{\mathcal{E}_1}\right]$$

and

$$\Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_0}\right] = \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}} = 1 \mid \overline{\mathcal{E}_0}\right] \cdot \Pr\nolimits_0\left[\overline{\mathcal{E}_0}\right] .$$

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

$$\Pr\nolimits_1\left[\overline{\mathcal{E}_1}\right] = \Pr\nolimits_0\left[\overline{\mathcal{E}_0}\right] .$$

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

$$\Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \mid \overline{\mathcal{E}_1}\right] \;=\; \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}} = 1 \mid \overline{\mathcal{E}_0}\right] .$$

Consequently,

$$\Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_1}\right] \;=\; \Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}0}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \overline{\mathcal{E}_0}\right] .$$

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

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

$$\Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1\right] \;\le\; \Pr\left[\mathbf{Exp}^{\mathrm{rsa}\text{-}\mathrm{ow}}_{\mathcal{K}_{\mathrm{rsa}},\mathcal{B}} = 1 \mid \overline{\mathsf{BAD}}\right]$$

and, by conditioning, that

$$\Pr\left[\mathbf{Exp}^{\mathrm{kr}\text{-}1}_{\mathcal{KR},\mathcal{A}} = 1 \wedge \mathcal{E}_1\right] \;\le\; \Pr\left[\mathbf{Exp}^{\mathrm{rsa}\text{-}\mathrm{ow}}_{\mathcal{K}_{\mathrm{rsa}},\mathcal{B}} = 1 \wedge \overline{\mathsf{BAD}}\right] \cdot \frac{1}{\Pr_2\left[\overline{\mathsf{BAD}}\right]}$$

where $\Pr_2[\,\cdot\,]$ denotes the probability over $\mathbf{Exp}^{\mathrm{rsa}\text{-}\mathrm{ow}}_{\mathcal{K}_{\mathrm{rsa}},\mathcal{B}}$. Equation (19) follows from the above equation and the fact that $\Pr_2\left[\overline{\mathsf{BAD}}\right] = 1/(q+1)$.

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