Graceful Service Degradation (or, How to Know your Payment is Late)

Alexandr Andoni*
MIT

Jessica Staddon
PARC

ABSTRACT

When distributing digital content over a broadcast channel it's often necessary to *revoke* users whose access privileges have expired, thus preventing them from recovering the content. This works well when users make a conscious decision to leave the system or have misbehaved, but numerous cases exist in which the revocation is in error and users are consequently left with the often onerous burden of getting reinstated. We introduce a gradual form of revocation that we call *service degradation* that enables the content distributor to provide "cues" to the user in the form of degraded system performance. The cues alert the user to their impending revocation and allow them to take the necessary action to remain in the system. Our protocols build on techniques for broadcast encryption and spam-fighting to provide the appropriate form of service for this previously ignored class of users.

Categories and Subject Descriptors

E.4 [**Coding And Information Theory**]: Formal methods of communication; H.3.5 [**Online Information Services**]: Commercial Services; H.3.7 [**Digital Libraries**]: Dissemination

General Terms

Security, Theory, Algorithms.

Keywords

Degradation scheme, broadcast encryption, revocation scheme, copyright protection, moderately-hard functions.

1. INTRODUCTION

Traditional approaches to distributing content over a broadcast channel (see, for example, [12, 4, 26]) allow for two types of users: *privileged* users who are authorized to receive the content, and *revoked* users who are not. These approaches do not allow for users who are authorized currently, but due to a recently overdue bill payment or expiring trial period, are in danger of losing that status. For such users it is desirable to provide access to the content coupled with "cues" that this access will be short-lived without some action on their part (e.g. bill payment or purchase). Indeed examples abound of users who have found the task of being reinstated after an expired trial period frustratingly difficult (see, for example, [11]).

One possibility for such cues are written reminders (e.g. messages sent by email or snail mail). There is an inherent delay in sending such reminders, however, and they are often ineffective for a number of reasons (e.g. misdelivery, incorrect sorting by mail filters, etc.). What's preferable is a method that "binds" the cue to the content so that through accessing the content the user experiences the cue and is thus warned of their impending status change. A binding is proposed in [2] for a different purpose. In [2], the content is broken into pieces and encrypted in such a way that revoked users users can recover some of the pieces but not enough to get the value of the content (e.g. video content is very "choppy" for such users). This technique is good for reducing communication overhead (since fewer users need to be truly revoked) but it doesn't serve our purposes as it causes an insurmountable obstacle to content access rather than an obstacle that is unavoidable, and thus provides a useful cue, but that can be overcome.[1]

We introduce protocols for the broadcast content distribution setting (e.g. pay-TV, online software) that provide appropriate service to all three types of users. Our enabling idea is to use broadcast techniques for revocation (e.g., [12, 21, 14] for the secure distribution of "hints" to what we term, *variably hard* functions (a generalization of the "moderately hard" functions of [1] and the "pricing functions" of [9]). Hints allow the users to recover the session key from the variably hard function, and then the content, which is subsequently broadcast encrypted under this session key. The type of hint received determines the amount of work a user must do to recover the session key, and thus whether or not the user is provided with a cue. That is, a high-quality hint is given to the privileged users, a lesser quality hint is provided to users who are near the end of their trial period

*Most of this research was conducted while this author was an intern at PARC.

[1]If a degraded user can modify their system so that the service degradation is unobtrusive we still view the system as successful because the act of modification demonstrates they experienced the cue. We do not intend to prevent degraded users from accessing the content, but rather to warn them that they may be prevented in the future.

EC'05, June 5–8, 2005, Vancouver, British Columbia, Canada.

or who are behind on their bills, and no hint is given to the revoked users (and so it is infeasible for revoked users to recover the content). We term our protocols *degradation* protocols to emphasize that they extend the notion of service revocation to service degradation.

Our protocols enable privacy-enhanced content access that integrates well with previously proposed approaches to secure digital content distribution. A traditional approach to content distribution is to require some form of user identification (e.g. log-in information) before releasing the content. This facilitates tracking of users and is potentially privacy-compromising. In our protocols, communication is effectively one-way: content is broadcast in encrypted form and so the distributor is unable to collect usage data. Further, our approach is a straightforward extension of existing methods for secure content distribution which may make adoption easier.

To summarize, we make the following contributions:

- A model of secure broadcast-channel services that allows for graceful service degradation based on the new notion of variably hard functions (Section 2) and a general technique for constructing secure degradation schemes (Section 3).
- A service degradation scheme specifically designed for broadcast-channel services that seek to enforce a trial period (e.g. online software). The protocol incurs constant communication overhead and constant user storage (Section 4.1).
- A service degradation scheme for any broadcast-channel service that employs a revocation scheme such as [21, 14] and incurs communication overhead and user storage costs that are on the same order as the underlying revocation scheme (Section 4.2).

OVERVIEW. This paper is organized as follows. We discuss related work in Section 1.1 and provide a model of service degradation and introduce our new tool, variably hard functions, in Section 2. In Section 3 we discuss the difference between degradation and revocation and provide a general technique for constructing degradation schemes. Section 4 contains our main constructions; in Section 4.1 we provide one specifically designed for trial period services and in Section 4.2 we provide a more general construction. In Section 5 we discuss simulation results and we conclude in Section 6. Appendix A presents a new stateful LKH-like revocation scheme that asymptotically outperforms LKH [5] in terms of communication. Appendix B presents a summary of the notation used in the paper.

1.1 Related Work

The study of secure content distribution over a broadcast channel is initiated in [12, 4] in which symmetric key cryptography is used to securely communicate information to dynamic sets of users. Since such protocols allow the content provider to prevent or revoke a user from accessing the content, they are often termed *revocation* protocols (a convention we follow here). Several extensions and improvements to [12, 4] have followed; we highlight a few that are particularly relevant to our work.

The idea of using secret sharing-based techniques (i.e. techniques similar to Shamir secret sharing [23]) is refined in [18, 24, 20, 17], in which protocols are provided that have the advantage of ensuring revoked users cannot recover the content through collusion. In Section 3 we show how the approach of [18] can be modified to yield degradation protocols.

A particular type of secret sharing-based protocol called a "subset-cover" protocol is introduced in [21] and refined in [14]. Subset-cover protocols are tree-based and provide desirable communication overhead, and storage costs; in particular, $O(r)$ communication overhead (where r is the number of revoked users) and user storage that is polylog in the total number of users. Such protocols can form the basis for the degradation protocol we introduce in Section 4.2.

A key component of all of our protocols is what we term *variably hard* functions. Such functions are a generalization of the "pricing functions" introduced in [9] and "moderately hard" functions in [1]. In [9] pricing functions enable email senders to provide easy-to-check proofs of computational effort, thus providing evidence that the email is not spam. The functions in [1] provide easily checked proofs of memory accesses rather than computational effort, as the authors argue memory access speeds are less likely to vary across machines than computational power. Other functions that enable such proofs of work are discussed in [15]. All these works are motivated by scenarios such as bulk mail (or spam) in which there is no need to enforce different work levels on different users, hence they don't consider the notion of graceful service degradation that is our focus. Informally speaking, our variably hard functions yield easily checked proofs that a certain, variable, number of *operations* have been executed, where an operation could be closely aligned with a computation, a memory-access, or some other process. The number of executed operations varies according to what the user knows about the function, and this is what allows our protocols to enforce fine-grained service degradation.

Finally, we note that degradation is related to the notion of differentiated services for the Internet (see, for example, [10]) in that both techniques are intended to increase overall customer satisfaction by enabling a vendor (of content or Internet service, respectively) to recognize more categories of users. However our setting is fundamentally different in that we enforce service degradation on users based on their account status while differentiated service protocols allow users to choose their level of service according to the user's budget constraints and/or resource needs.

2. MODEL AND TOOLS

The familiar setting for broadcast-channel services (see, for example [12]) includes a center $\mathcal{C}$ that distributes content to a set, $\mathcal{N}$, of users. In each of a sequence of *sessions*, encrypted data is broadcast. Each broadcast includes a header $Head$, consisting of an encrypted session key, SK, and the content encrypted under this session key. $\mathcal{C}$ can encrypt the content in such a way that a subset $P \subseteq \mathcal{N}$ of "privileged" users can decrypt the content, while the the users in the set $R = \mathcal{N} \setminus P$, are not able to recover the content and are thus termed "revoked". The revoked users are those who have not paid for the content. The sets $\mathcal{N}, P, R$ can change over time. Schemes for distributing broadcast-channel services to the dynamically changing sets, $\mathcal{N}, P$ and R are often called *revocation* schemes.

We consider an extension of this setting that we term broadcast-channel services with degradation, or simply, a *degradation scheme*, in which the set of users $\mathcal{N}$ is partitioned into *three* subsets $\mathcal{N} = P \cup R \cup D$. The sets P and R have the same roles as before, while the new set D denotes the users who experience degraded service in terms of the ease with which they recover the content from the broadcast and hence we term these users "degraded". A user in D can decrypt SK and the content, but doing so requires additional "work" (much more than for a user in P). The amount of work can be varied as $\mathcal{C}$ desires and may be measured computationally [15], by memory accesses [8] or by any other operation that is deemed to not advantage any particular user. We enforce this work requirement by the use of a "hard function", $F(\cdot)$, (defined precisely in Section 2.1) whose value on input v determines session key, SK. Informally speaking, computing $F(v)$ requires a "hint"; the amount of hint that is available is inversely correlated with the amount of work that's expended when generating $F(v)$. In particular, users in R are unable to recover a hint from the header, and so it is infeasible for such users to recover SK, while users in D recover partial hints from the header and thus are able to recover SK with some work, and users in P recover a "complete" hint (i.e. v) and thus reconstruct SK efficiently. Indeed, we require that a degradation scheme be able to vary the amount of work a user in D must perform according the the *class* the user is in, where the class indicates how severely the user is to be degraded. The class of a user may be a function of when the user last paid for the content. We induce an ordering on the classes such that users in P are thus in the "highest" class, whereas users in R are in the "lowest" class. the following definition makes this precise.

DEFINITION 1. *Let $\mathcal{L} = \{C_0, \ldots, C_{L+1}\}$ be the set of all classes, where $P = C_0$, $R = C_{L+1}$ and $D = \cup_{j=1}^{L} C_j$. We order the classes as $P = C_0 > C_1 > \ldots > C_{L+1} = R$ and say a class C_{j_1} is* higher than *C_{j_2} if $C_{j_1} > C_{j_2}$ according to this ordering. In each session j, there exists a class determining function $h_j : \mathcal{N} \to \mathcal{L}$ called the* user hierarchy *for that session. In the event that user transitions between classes are restricted, we denote the set of allowed hierarchy transitions (i.e., allowed pairs (h_{j-1}, h_j)) by $\mathcal{T} \subseteq \mathcal{F} \times \mathcal{F}$, where $\mathcal{F}$ is the set of all functions $h : \mathcal{N} \to \mathcal{L}$.*

Before defining a degradation scheme we introduce some terminology and notation. Let κ be the security parameter (the width of the keys). We say $\eta : \mathcal{R} \to [0, 1]$ is a *negligible* function if for any $c \in \mathcal{R}$, there exists $\kappa_c \in \mathcal{R}$, such that $\eta(\kappa) < 1/\kappa^c$ for all $\kappa \geq \kappa_c$.

We use $x \in_r S$ to indicate that the element x is chosen uniformly at random from the set S.

Lastly, we often make use of a symmetric encryption scheme $E_k(\cdot)$, that is assumed to be "perfect" in the sense that, without knowing the key k, it is infeasible to decrypt a ciphertext, $E_k(M)$, or deduce any information on the key k. In Section 4.2, given a set $S \subseteq \mathcal{N}$ we use the notation, $\mathcal{E}^S(M)$ to denote the use of a revocation scheme (e.g. [21, 14]) to encrypt a message, M so that users in S are each able to recover M and users who are not in S are unable to, even when colluding. $\mathcal{E}^S(M)$ might actually consist of a set of ciphertexts generated with a symmetric encryption scheme under various keys (e.g. $\mathcal{E}^S(M) = \{E_{k_1}(M), \ldots, E_{k_j}(M)\}$) according to the specifics of the revocation scheme.

DEFINITION 2. *A* degradation scheme *uses the following four algorithms, the first three of which are randomized:*

1. *A parameter generation algorithm,* $\mathsf{Params}(1^\kappa)$*, that takes as input the security parameter κ and outputs the pair of system parameters (σ, ρ). σ are the private (secret) parameters and ρ are public parameters.*
2. *A key generation algorithm,* $\mathsf{KeyGen}(\rho, u)$*, that outputs the private key K_u for user u. User u will use K_u to recover the session keys from the headers.*
3. *A header generation algorithm,* $\mathsf{Encrypt}(j, \sigma, \rho, h_{j-1}, h_j, SK_i)$*, that outputs the broadcast header $Head_j$ for session j. The goal of the broadcast header $Head_j$ is to distribute the session key SK_j. h_{j-1} and h_j are the old and respectively the new hierarchies of users.*[2]
4. *A session key recovery algorithm,* $\mathsf{KeyRecovery}(\rho, K_u, B_j)$*, that generates the session key SK_j given the key K_u of user u.*

In addition, a degradation scheme satisfies the following properties:

- *(Correctness)*
  1. *If $u \in P$ in hierarchy h_j, then* $\mathsf{KeyRecovery}(\rho, K_u, B_j)$ *recovers SK_j in a constant number of operations*[3]*, where $B_j =$* $\mathsf{Encrypt}(t, \sigma, \rho, h_{j-1}, h_j, SK_j)$*;*
  2. *If $u \in R$ in hierarchy h_j, then u cannot recover SK_j from B_j with non-negligible probability (in a number of operations polynomial in κ);*
  3. *If $u \in C_i$ in hierarchy h_j, then* $\mathsf{KeyRecovery}(\rho, K_u, B_j)$ *recovers SK_j with $O(UT_i)$ operations but cannot recover SK_j in $O(LT_i)$ operations with probability greater than $\Omega\left(\frac{1}{LT_i}\right)$, where UT_i, LT_i are given limits;*
- *(Collusion resistance for revoked users) In any session j, for any set $S \subseteq R$, it is computationally infeasible for the coalition of users in S to recover SK_j.*

2.1 Variably Hard Functions

In this section we formally define the functions that our degradation schemes rely upon. The following definition is more general, and somewhat more formal, than the notions of "pricing" functions in [9, 8] and "moderately hard" functions in [1].

DEFINITION 3. *Let $v \in_r \{0,1\}^\kappa$. Let $g : \{0,1\}^\kappa \to \mathcal{D}$ be a function, for some domain $\mathcal{D}$ that is injective with high probability. We call $g(\cdot)$ a* test function*. Let*

$$Hints(v) \subseteq \left\{ Y^{(\ell)} \subseteq \{0,1\}^\kappa \mid v \in Y^{(\ell)}, |Y^{(\ell)}| = 2^\ell, 0 \leq \ell \leq \kappa \right\}$$

be the set of hints *for $v \in \{0,1\}^\kappa$ (i.e., a hint for v is a subspace $Y^{(\ell)}$ of $\{0,1\}^\kappa$ of size 2^ℓ that contains v). A function, $F : \{0,1\}^\kappa \to \{0,1\}^\kappa$, is a* variably hard function *if the following hold:*

1. *For any given $v \in_r \{0,1\}^\kappa$, $g(v)$ and $F(v)$ are both efficiently computable with a polynomial in κ number of operations;*
2. *There exist constants c, ℓ_0 such that for $v \in_r \{0,1\}^\kappa$, and any $\ell \geq \ell_0$, any algorithm performing $O(2^{c\ell})$ operations can generate $F(v)$ given only $g(v)$ and $Y^{(\ell)} \in Hints(v)$ with probability at most $O(2^{-c\ell})$.*

[2]Note that h_{j-1} is only required for the construction of Section 4.2 but we include it in the definition for completeness.

[3]We regard the decryption of a ciphertext $E_k(M)$ to take a constant number of operations. Such a decryption is all a privileged user must do to recover the content.

We use a variably hard function F, and a corresponding test function g, in the following way in degradation schemes. F and g are made public. The center, $\mathcal{C}$, generates a random value $v \in_r \{0,1\}^k$ and encrypts some content with the key derived from $F(v)$. Further, the center wants to allow a user $u \in D$ to decrypt the content, but only after u performs $O(2^\ell)$ operations. For this purpose, the center gives $g(v)$ and some $Y^{(\ell)} \in Hints(v)$ to u. Thus, assuming a perfect encryption scheme, to decrypt the content, u needs to compute $F(v)$; and doing so requires u to perform at least $\Omega(2^{c\ell})$ operations (as guaranteed by the definition).

Intuitively, the hardness of F relies on the hardness of recovering v from $g(v)$ and $Y^{(\ell)}$. That is, the one-wayness of $g(\cdot)$, even given Y^ℓ, is necessary.

Note that a user can compute $F(v)$ given $g(v)$ and $Y^{(\ell)}$ with at most $O(2^\ell \cdot \mathrm{poly}(\kappa))$ operations, where $\mathrm{poly}(\kappa)$ is a polynomial in κ. One way to accomplish this is by enumerating all elements $y \in Y^{(\ell)}$ and testing whether $g(y) = g(v)$; once $\mathcal{A}$ finds such y, $F(y)$ is easily computed and $F(y) = F(v)$. Since $|Y^{(\ell)}| = 2^\ell$, this process incurs $O(2^\ell \cdot \mathrm{poly}(\kappa))$ operations. In the rest of the paper, we will usually drop the $\mathrm{poly}(\kappa)$ factor since the exponential is much more important. Note that saying that the user can compute $F(v)$ with $O(2^\ell)$ effort is not in conflict with the second condition of the Definition 3 even when $c = 1$: the two $O(\cdot)$ notations denoting the operational effort hide different constants. [4]

2.1.1 Variably Hard Function Constructions

In this section we construct two variably hard functions. The first is based on computational hardness and the second on memory-access hardness, the latter is considered a more uniform across different platforms [1].

A Variably Hard Function from a One-Way Permutation. A natural candidate for a variably hard function is a one-way permutation. Let P be a one-way permutation on $\{0,1\}^\kappa$ that for which any inverter running in time $O(2^{c\kappa})$, succeeds with probability $O(2^{c\kappa})$, where c is a constant $c \leq 1/2$. Then, define the test function $g(v) = P(v)$, and $F(v) = v$, for all $v \in \{0,1\}^\kappa$. . We claim $F(\cdot)$, with test function $g(\cdot)$, is a variably hard function, since computing $F(v)$ given $g(v)$ and $Y^{(\ell)} \in Hints(v)$, requires at least $\Omega(2^{c\ell})$ computation (one can prove this via a standard simulation/reduction argument). On the other hand, one can compute $F(v)$ with $O(2^\ell)$ computation by trying all possible $v \in Y^{(\ell)}$.

A deficiency of this function is that it requires a fixed amount of computation – resulting in different times on different processors. Ideally, we would like a function that imposes the same time on all platforms, independent of the processing speed. To address this deficiency, we propose the next hard function, which requires fixed amount of memory accesses rather than a fixed number of CPU cycles. The benefit of this approach is that memory accesses take an amount of time that is more uniform across different platforms (as argued by [8, 1]).

A Variably Hard Function Based on [8]. Next we construct a hard function that requires a fixed amount of memory accesses as opposed to requiring a fixed amount of CPU cycles. For this purpose, we use a modification of the memory-bound functions that are used for fighting spam in [8].

We first recall the memory-bound function construction in [8], and then describe how we modify the function to construct a variably hard function. The input to the memory-bound function is a tuple, r, containing the message of the email, the receiver, and the date. The output of the function is a value z for which the procedure `Test(`r`, `z`)` described below succeeds. `Test(`r`, `z`)` employs four hash functions, $H_0(\cdot), H_1(\cdot), H_3(\cdot), H_4(\cdot)$, and a fixed table T. It is parameterized by positive integers t and ℓ, where ℓ determines the "hardness" of the function. The definition of `Test(`r`, `z`)` follows:

`Test(`r`, `z`):`
  $A \leftarrow H_0(r, z)$
  `For` $i = 1 \ldots t$:
    $c_i \leftarrow H_1(A)$
    $A \leftarrow H_2(A, T[c_i])$
  `Succeed if, after the loop above, the last` ℓ `bits of` $H_3(A)$ `are all zero.`
  `Fail otherwise.`

In [8], the authors prove that in order to compute this function, one needs $\Omega(2^\ell t)$ memory accesses to T (for certain specified assumptions on the parameters and the platform).

The above memory-bound function must be modified in order to yield a variably hard function because it is in fact a one-to-many relation since there might be several values z for which `Test(`r`, `z`)` succeeds. This is not an issue for spam-fighting, however this is at odds with our goal of ensuring each user generates the same value (which is, or can be used to generate, the session key). In addition, to allow for many classes of degraded users we need to be able to parameterize the number of operations required of a user to recover this common value. To achieve these properties we define the function $U(r, z)$, $r \in \{0,1\}^\kappa, z \in \{0,1\}^\kappa$, as follows:

  $A_0 = H_0(r, z)$
  `For` $i = 1, \ldots t$:
    $c_i = H_1(A_{i-1})$
    $A_i = H_2(A_{i-1}, T[c_i])$
  $U(r, z) = H_3(A_t)$

In the terminology of the function U, note that the original function of [8] requires finding a z^* for which the last ℓ digits of $U(r, z^*)$ are zero; z^* is then the output of the function from [8]. We don't make this requirement but rather simply generate $H_3(A_t)$ as the output.

To make this into a *variably hard* function, we define the test function as $g(v) = \langle H_4(v), U(H_4(v), v)\rangle$, where $H_4(v)$ is a hash function with codomain $\{0,1\}^\kappa$ (modelled as a random oracle), and we let $F(\cdot)$ be the identity function (i.e. $F(v) = v$ for all $v \in \{0,1\}^\kappa$.

To see that $F(\cdot)$, with the test function $g(\cdot)$, is a variably hard function note first that $g(\cdot)$ and $F(\cdot)$ are efficiently computable. Furthermore, since $F(v) = v$, computing $F(v)$ is the same as inverting $g(v) = \langle H_4(v), U(H_4(v), v)\rangle$, or, equivalently, finding z^* such that $U(H_4(z^*), z^*) = U(H_4(v), v)$. Given a hint $Y^{(\ell)} \in Hints(v)$, finding such $z^* \in Y^{(\ell)}$ takes, in expectation, $\Omega(2^\ell t)$ memory accesses by the same argu-

[4] There is a technicality relating to the hints $Y^{(\ell)}$: it can take roughly 2^ℓ bits to describe $Y^{(\ell)}$. This is infeasible since it would imply 2^ℓ communication and a similar effort on center's part. To resolve this technicality, we restrict the set $Hints(v)$ to sets $Y^{(\ell)}$ that have a short description. In particular, for the purpose of this paper, it is sufficient to consider only the sets $Y^{(\ell)}$, where a set $Y^{(\ell)}$ is the set of solutions $y \in \{0,1\}^\kappa$ to the equation $Ay = b$, where $A \in M_{\kappa-\ell,\kappa}[\mathcal{Z}_2]$ is a matrix of rank $\kappa - \ell$ and $b \in \mathcal{Z}_2^{\kappa-\ell}$. In this case we can describe $Y^{(\ell)}$ with only $O(\kappa^2)$ bits.

ment as that involved in the proof of Theorem 1 of [8].

Lastly, we mention that we need the same assumptions on the architecture and parameters $|A|, t, h$ as in Theorem 1 of [8], as well as the additional assumption that the codomain of $U(\cdot)$ is $\{0,1\}^{2\kappa}$ (2 has no particular importance and can be replaced by other constant bigger than 1). Thus, there is a negligible probability of the existence of two different $v', v'' \in \{0,1\}^{\kappa}$ such that $U(H_4(v'), v') = U(H_4(v''), v'')$, and thus each user will arrive at the same output that in turn can be used to generate the same session key, SK.

3. DEGRADATION VS. REVOCATION

n this section we discuss the difference between degradation and revocation and provide an illustrative construction. Since degradation is a generalization of revocation, any degradation scheme yields a revocation scheme. Conversely, a revocation scheme that is capable of revoking sufficiently many users, coupled with a hard function, can be used to accomplish degradation. This is done by repeatedly invoking the revocation scheme to target users in the various degradation classes. For example, in the case of a single degradation class, let $v = (v_1, \ldots, v_\kappa)$ be such that the session key is (or, is derived from) $F(v)$, for some variably hard function, $F(\cdot)$. The revocation scheme can first be used to generate an encrypted broadcast from which the users in P can recover $v_1, \ldots, v_\ell$ for some $\ell < \kappa$, and then the revocation scheme can be used again to generate an encrypted broadcast from which the users in $P \cup D$ can recover $v_{\ell+1}, \ldots, v_\kappa$. The "check" value $g(v)$ can be broadcast unencrypted. After these three broadcasts, users in P have the value v (the "complete" hint) and thus can easily recover the session key. Users in D have a partial hint (indeed, an ℓ-dimensional subspace that contains v) and can recover the session key in $O(2^\ell)$ operations (for example, by repeatedly guessing the v and testing each guess against $g(v)$). Finally, it is infeasible for users in R to recover the session key because doing so requires guessing a bit string of length κ. This technique can be extended to more than three classes while incurring a number of broadcasts that is proportional to the number of classes. We call this technique for using a revocation scheme to degrade users *repeated revocation*.

Because repeated revocation requires a revocation scheme capable of revoking $|D| + |R|$ users (rather than just $|R|$) it may be less efficient. That is, given the constraints of the particular application, it may be desirable to construct a degradation scheme *directly* rather than via a revocation scheme. In this section, we demonstrate one such construction inspired by the revocation protocol of [18].

Before describing our construction we give a brief overview of the structure of our degradation protocols (presented in this section and section 4). Recall that we have $L+2$ classes, $C_0, \ldots C_{L+1}$, with the convention that users from the class C_i are "degraded more" than the users from the class C_{i-1}. The union of $C_1, \ldots C_L$ is the set of all degraded users, D. To each class C_i, we assign a hardness parameter, e_i. For users in class C_i, it will take roughly 2^{e_i} operations to compute $F(v)$ for some input, v. For $i = 2, \ldots, L$, the fact that $e_{i-1} \leq e_i$ ensures that users in C_i have no better service than users in C_{i-1}.

Finally, we make use of a variably hard function, F, in the following basic way. Consider some value $v \in \{0,1\}^\kappa$ and its associated set of partial hints, $Hints(v)$. We ensure that each user in C_i is able to recover an e_i-dimensional space, $Y^{(e_i)} \in Hints(v)$. By the definition of $F(\cdot)$, a user in C_i ($i > 1$) has to reduce the space of possible values for v in order to compute $F(v)$, and doing so takes approximately $O(2^{e_i})$ operations. Note that all users from $P \cup D = C_0 \cup \ldots \cup C_L$ will compute the same value $F(v)$.

CONSTRUCTION. For simplicity of exposition we assume a single degradation class ($L = 1$), that is, all the users in D experience the same impaired level of service. Our construction is similar to the construction in [18] but differs in the definition of the session key and the $\mathsf{Encrypt}(\cdot)$ algorithm. In the following, let $0 < p < 1$ be a constant chosen by the center, $\mathcal{C}$, and let t, γ be positive integers such that $(t+1)\gamma = \kappa$.

1. Params outputs m keys, $k_1, \ldots, k_m \in \{0,1\}^\kappa$, a randomly generated polynomial $f(x) = a_t x^t + \ldots + a_0 \in F_{2^\gamma}[x]$, and $\mathcal{C}$ partitions the keys into r sets, $S_1, \ldots, S_r$, where for $i = 1, \ldots, r$, $S_i = \{k_{i,1}, \ldots, k_{i,s}\} \subseteq \{k_1, \ldots, k_m\}$, for some integer, $s > 0$.
2. For every $u \in \mathcal{N}$, KeyGen allocates to u a randomly chosen key from each of $S_1, \ldots, S_r$. Thus, a user u stores r keys in total.
3. $\mathsf{Encrypt}$ outputs the pairs, $\{(b_k, E_k(f(b_k))) : b_k \in_r F_{2^\gamma}, k \in T \subseteq \{k_1, \ldots, k_m\}\} \cup \{g(a_t || \ldots || a_1 || a_0)\}$, where $g(\cdot)$ is a test function for some variably hard function $F(\cdot)$ (Definition 3), and the set T is chosen from $\{k_1, \ldots, k_m\}$ as follows: for $i = 1, \ldots, m$, $k_i \in T$ with probability α where,
   - $\alpha = 0$ if there exists $u \in R$ such that u received k_i from KeyGen,
   - $\alpha = p$ if for every $u \in R$, u did not receive k_i from KeyGen but there exists a user $u \in D$ who received k_i from KeyGen,
   - Otherwise, $\alpha = 1$.

To intuitively see that this construction is a degradation protocol note that a user in P is more likely to have a key, $k \in T$ than a user in D. Further, a user in R has none of the keys in T and hence, a user in D is more likely to have a key, $k \in T$ than a user in R. The size of the subsets of $\{(b_k, f(b_k)) | k \in T\}$ that are recovered by users during the invocation of $\mathsf{KeyRecovery}$ are consequently expected to be of decreasing size for users in P, D and R respectively. These subsets are essentially hints for $v = (a_t || \ldots || a_0)$, and hence privileged users have more substantial hints than degraded users, who in turn have more substantial hints than revoked users. We provide an example lemma showing how the parameters can be chosen to achieve different operation costs for the three types of users.

LEMMA 1. *Let $p \leq \frac{(1-1/s)^{|D|}}{2+(1-1/s)^{|D|}}$. In the degradation protocol given above the parameters can be chosen so that the following hold:*

1. *A user in P is expected to have at least $3/2(t+1)$ keys in T and with probability more than $1 - 1/e^{(t+1)/12}$ can recover $(a_t || \ldots || a_1 || a_0)$ during* $\mathsf{KeyRecovery}$*, and thus, SK, in constant time.*
2. *A user in D is expected to have $(t+1)/2$ keys in T and with probability at least $(1 - 1/e^{(t+1)/24})(1 - 1/e^{(t+1)/16})$ such a user incurs an operational cost of at least $\Omega(2^{\gamma t/4})$ and at most $O(2^{3\gamma t/4})$, to recover SK.*
3. *A user in R has no keys in T and so it is infeasible for such a user to recover SK.*

In addition, to achieve these properties with a repeated revocation protocol using underlying revocation scheme [18], *one incurs a comparable user storage costs but a factor of* $\frac{1+(1-1/s)^{|D|}}{(1-p)(1-1/s)^{|D|}+p} > 1$ *more communication overhead.*

PROOF. To verify the claims on users in P for the construction above, note that for a key k allocated to a user in P, the probability that k is in T is, $(1-1/s)^{|R|}[p+(1-p)(1-1/s)^{|D|}]$, and so the expected number of keys that a privileged user has in T is, $\mu = r(1-1/s)^{|R|}[p+(1-p)(1-1/s)^{|D|}]$. Setting $\mu > 3/2(t+1)$, Chernoff bounds (see, for example, [19]) give the probability in statement 1. of the lemma. Note that this implies the following lower bound on r, $r \geq \frac{3(t+1)}{2(1-1/s)^{|R|}[p+(1-p)(1-1/s)^{|D|}]}$.

To verify the claims on users in D in the construction above, note that for a key k allocated to a user in D, the probability that k is in T is, $(1-1/s)^{|R|}p$, and so the expected number of keys that a degraded user has in T is $\mu = (1-1/s)^{|R|}pr$. Finally, Chernoff bounds yield the probability stated in 2. Finally, by setting $\mu = (t+1)/2$ we get the following equation, $r = \frac{t+1}{2p(1-1/s)^{|R|}}$.

Since T is constructed by discarding any key known to a user in R, statement 3. of the lemma holds.

Hence, choosing parameter values so that $r = \frac{t+1}{2p(1-1/s)^{|R|}} \geq \frac{3(t+1)}{2(1-1/s)^{|R|}[p+(1-p)(1-1/s)^{|D|}]}$ yields a construction satisfying 1-3 above, and these two conditions hold if and only if $p \leq \frac{(1-1/s)^{|D|}}{2+(1-1/s)^{|D|}}$. Substituting the upper bound on p in the expression for r, we get the following lower bound on r: $r = \frac{t+1}{2p(1-1/s)^{|R|}} \geq \frac{(t+1)(2+(1-1/s)^{|D|})}{2(1-1/s)^{|R|+|D|}}$.

To compare repeated revocation and our degradation construction we first recall how repeated revocation works. We first form a set T_1, by discarding all keys known to users in $D \cup R$, and then broadcast a set of pairs $\{(b_k, E_k(f(b_k))|k \in T_1\}$. For the second broadcast we form a set T_2 by discarding all the keys known to users in R, and we broadcast a set of pairs $\{(a_{k'}, E_{k'}(f(a_{k'}))|k' \in T_2\}$. We choose $b_k, a_{k'} \in F_{2^\gamma}$ all distinct.

Privileged users must recover the most points on $f(\cdot)$ from the broadcast and hence, by considering a privileged user we get the tightest lower bounds on user storage. In order to make a fair comparison with our degradation protocol we require a privileged user to recover at least $3(t+1)/2$ points on $f(\cdot)$ from the broadcasts in expectation. Since degraded users and privileged users are each expected to recover $(t+1)/2$ points from the second broadcast this implies a privileged user is expected to recover more than $t+1$ points from the first broadcast, and equivalently, $r(1-1/s)^{|R|+|D|} \geq t+1$. This implies $r \geq \frac{t+1}{(1-1/s)^{|R|+|D|}}$. Hence, user storage is comparable in the two protocols (recall, our lower bound on r is $\frac{(t+1)(2+(1-1/s)^{|D|})}{2(1-1/s)^{|R|+|D|}}$). The expected communication overhead for repeated revocation is roughly $|T_1| + |T_2|$, or $m(1-1/s)^{|R|}(1+(1-1/s)^{|D|})$ since there are m keys in total. For the degradation protocol, the expected size of T is $m(1-1/s)^{|R|}[(1-1/s)^{|D|} + p(1-(1-1/s)^{|D|})]$, which is smaller than in the repeated revocation case by a factor of, $\frac{(1-p)(1-1/s)^{|D|}+p}{1+(1-1/s)^{|D|}}$.

$\square$

A significant difference between this example construction and the repeated revocation approach is that this example provides no collusion resistance against *degraded* users (although, as in repeated revocation, *revoked* users gain nothing by colluding). It is certainly possible to vary the parameters to gain some collusion resistance against degraded users, however we point out that the purpose of the degradation scheme is to alert or warn degraded users that they may be revoked soon without action on their part, and *not* to prevent them from accessing the content. Hence, collusion resistance seems less important in their case.

This construction shows that it is possible to design degradation protocols that are better than what an obvious use of revocation, that is the repeated revocation protocol, provides. In the following section we explore this further and provide two degradation protocols tailored to specific online service settings.

4. USAGE-TAILORED DEGRADATION

In this section, we present two different degradation schemes, each tailored to a different usage scenario. The schemes are designed to meet the constraints of each usage setting, often leveraging the specific attributes of the usage scenario to do so. For example, in the first construction we assume the degradation schedule is known in advance, as is the case with online services that offer a trial period to new users. Online training (see for example, [25, 6]), online consumer applications (see, for example, [16]) and pay TV are three markets in which trial periods are often given. In such a setting, users have access to the online service for a trial period (e.g., of 30 days); after the trial period ends, users are still able to use the service, but with degraded quality. The level of degradation depends on the time elapsed since the end of the trial period and takes the form of a delay in the user's ability to access the service, *not* in the quality of the service once it is received. In other words, a trial period is followed by a degradation period that gently reminds users that the trial period has ended. Because the degradation schedule is known in advance we are able to meet it with constant communication overhead and user storage.

In the second construction, we allow for an unpredictable degradation schedule, as is most likely in the subscription mode of an online service. Users subscribe to the service and payments are due at regular intervals (e.g., monthly). With a degradation scheme for subscriptions, when a user is late on the payments (an event that is unpredictable by the service provider), the user experiences a delay in accessing the service, with the exact amount of delay depending on the overdue period. In this way, the user is reminded to pay the overdue bill. Because this setting is similar to the conventional revocation setting in that the class of a user can change in an unpredictable manner, we are able to leverage existing revocation techniques to achieve our degradation goals with communication overhead and user storage that are on the same order as what is incurred by the revocation techniques.

Finally, we note that both of our constructions achieve collusion resistance that is stronger than what is required by Definition 2. In particular, not only are users in R unable to access the service content through collusion, but users in D are unable to reduce the total amount of work needed to retrieve the content through collusion. Although we don't view this as a crucial property of a degradation scheme (since the sole purpose of the work enforced on users in D is to provide a warning that they may soon be revoked) it is desirable as it makes the hierarchy robust.

4.1 Known degradation schedule

The following is a precise description of the service schedule for a user who is granted access to some online service for a trial period of T days. Recall that we view the service as being made available in a sequence of sessions. Assuming a user signs up for the service on day t, the user is allowed to decrypt broadcast sessions with essentially no operational cost[5] on days $t, t+1, t+2, \dots, t+T-1$ (i.e. during this time the user is in set P). On day $t+T$, the user needs 2^{e_1} time to decrypt a session (i.e. the user is transferred to class C_1); in general, for $i = 1 \dots L$, on day $t+T-1+i$, the user is in class C_i and needs 2^{e_i} operations to decrypt a session. On day $t+T+L$, the user is in set R, that is the user cannot decrypt any session at all.

To implement this policy we define the following scheme in which the initialization algorithm, $\mathsf{Params}(1^\kappa)$, establishes:

- A variably hard function $F(\cdot)$ with a corresponding test function $g(\cdot)$;
- A one-way permutation $W : \{0,1\}^\kappa \to \{0,1\}^\kappa$, such that any algorithm running in $O(2^{\zeta\kappa})$ inverts W with probability $O(2^{-\zeta\kappa})$, for some constant $\zeta \le 1$;
- $\delta_i = e_i - e_{i-1}$ (with the convention that $e_0 = 0$ and $e_{L+1} = \kappa$);
- Sets $\{K_t\}_t$ and $\{A_t\}_t$ such that $K_t = W(K_{t+1})$ and $A_t = W(A_{t+1})$, $t \ge 1$; these are formed by randomly choosing values $K_{t+\alpha}, A_{t+\alpha}$, where α is a big integer (e.g., representing several years), and computing respectively K_t, A_t.[6]

The key distribution, KeyGen, works as follows. If a user u began a trial period on day t, they get K_{t+T-1} and A_{t+T-1}. This allows the user u to decrypt the content without additional work in sessions on days $\{t, \dots, t+T-1\}$, and decrypt as the expense of additional work on days $\{t+T, \dots, t+T+L-1\}$.

On any given day there are several sessions (the number of the sessions is a function of how many times we want to impose the work overhead to the degraded users). The broadcast made at the beginning of a session is composed of the header, $Head$, that contains the encrypted session key, SK. The content is then broadcast encrypted under the session key, SK.

The header encrypting the session key has the structure outlined in Figure 4.1.

The decryption algorithm works as follows. Suppose the user subscribes on day t_0. On day t, $t < t_0 + T$ (within the trial period), the user can simply compute $K_t = W^{t_0+T-t-1}$ (K_{t_0+T-1}), and decrypt $E_{K_t}(SK)$. When $t_0 + T \le t < t_0 + T + L$, the user is in class $C_i = C_{t-t_0-T+1}$ because the user knows only the first $\kappa - e_i$ bits of v, which are $y = \langle G_{\delta_{L+1}}(M \oplus A_{t-L}), \dots, G_{\delta_{t-t_0-T+2}}(M \oplus A_{t_0+T-1})\rangle$. Thus, the user has the hint $Y^{(e_i)} = \{y\} \times \{0,1\}^{e_i}$, with which the user can compute $F(v)$ with $O(2^{e_i})$ operational effort (as mentioned in 2.1, this is not in conflict with definition 3). Knowing $F(v)$, the user decrypts $E_{F(v)}(SK)$ to recover SK.

4.1.1 Security Proof

Next we prove that the construction above indeed satisfies the properties of a degradation scheme according to Definition 2. Note that the first correctness property is trivially satisfied as discussed above.

Since revoked users know none of the κ bits of v, the scheme is collusion resistant. Also, note that the scheme is resistant to collusion by *degraded* users (as discussed in the introduction to Section 4) because of the manner in which K_t and A_t are distributed: for any set S of users, the user who subscribed most recently is able to compute all the information known by everybody in S. Thus, in a coalition of users, the user in the best class does not learn anything new and cannot raise their class.

Next, we prove second and third correctness properties are satisfied with $UT_i = 2^{e_i}$ and $LT_i = 2^{ce_i}$, where c is the constant from the Definition 3 of the hard functions. Since a user can compute $F(v)$ with $O(2^{e_i}) = O(UT_i)$ effort, we need only to prove that a user in class C_i, $i = 1 \dots L+1$, cannot recover SK with operational effort of $O(2^{ce_i})$ and probability $\Omega(2^{-ce_i})$. Recall that a user in C_i began their trial period on day $t - T - i + 1$ at which time they were given K_{t-i}, A_{t-i}, from which they can compute, $\{K_{t-i}, K_{t-i-1}, \dots, K_1,\ A_{t-i},\ A_{t-i-1}, \dots,\ A_1\}$. Since $E(\cdot)$ is a perfect encryption, the user needs to recover either K_t or $F(v)$, and decrypt the corresponding part of the header, in order to recover SK. The user cannot compute K_t due to the one-wayness of the permutation W.

The main part of the proof is showing that a user cannot compute $F(v)$ with probability greater than $\Omega(2^{-ce_i})$ at an operational expense of $O(2^{ce_i})$. The intuition behind the proof is that if the user can compute $F(v)$ fast, then either 1) the user, who is a member of class C_i, can compute A_t's that are not in the set $\{A_{t-i}, A_{t-i-1}, \dots, A_1\}$ and thus can invert $W(\cdot)$ fast; or 2) the user computes $F(\cdot)$ faster than specified by the definition of the variably hard function. To show this, we consider a user algorithm, $\mathcal{A}$, for a user in class C_i. Note by definition of variably hard functions, in order to compute $F(v)$, $\mathcal{A}$ must compute additional bits of v and doing so requires the the output of some random oracles G_{δ_j} on inputs that include some $A_j \notin \{A_{t-i}, A_{t-i-1}, \dots, A_1\}$. If $\mathcal{A}$ ever queries G_{δ_j} on input that includes such an A_j (which we refer to as a *successful* call), then this means $\mathcal{A}$ was able to invert W. If no successful calls were ever issued, then $\mathcal{A}$ knows only the hint $Y^{(e_i)}$ and, therefore, should not be able to compute $F(v)$ faster than specified by the definition. A formal argument follows.

Consider a randomized algorithm $\mathcal{A}$ that makes $\tau = O(2^{ce_i})$ operational effort and generates $F(v)$ with probability π. In what follows, we show that $\pi \le O(2^{-ce_i})$, under the assumptions that $ce_i < \zeta\kappa/2$ and that $\mathcal{A}$ runs in time $O(2^{\zeta\kappa})$ (easily satisfied by the appropriate choice of κ).

The input to $\mathcal{A}$ comprises: information learned at subscription time (A_{t-i}); information learned from the current header $Head^{known}$ $(M, g(v))$; and information learned from the previous broadcasts $(M_{f,s}, G_{\delta_j}(M_{f,s} \oplus A_{f-j+1})$, for $t-i+1 \le f \le t$, $1 \le j \le f-t+i$, $1 \le s \le n_{sns}$, where n_{sns} is the maximum number of sessions per day). All other information can be computed from the above information.

During $\mathcal{A}$'s execution, $\mathcal{A}$ can generate several calls to the oracles G_{δ_j}. Again, we say a call to oracle G_{δ_j} is *successful* if it is made with input $M_{f,s} \oplus A_{f-j+1}$ for some valid f, j, s. For technical reasons, we are interested in the case when all $M_{f,s} \oplus A_{f-j+1}$ are different, which happens with probability at least $1 - \frac{L^2}{2^{\zeta\kappa}} - \frac{n_{sns}^2 L^4}{2^\kappa}$. This probability ex-

[5]Recall, we regard the cost of decrypting a ciphertext $E_k(M)$ when k is known, to take a constant number of operations.

[6]In fact, to reduce the cost of computing K_t and A_t from $K_{t+\alpha}$ and $A_{t+\alpha}$, we can modify slightly the scheme, for example, to use two sets of sequences each with $\alpha = L$.

$$Head^{known} = \langle t, M, g(v), E_{K_t}(SK), E_{F(v)}(SK) \rangle$$

- $M \in_r \{0,1\}^\kappa$,
- $v = \langle\ G_{\delta_{L+1}}(M \oplus A_{t-L}), G_{\delta_L}(M \oplus A_{t-L+1}), \ldots, G_{\delta_2}(M \oplus A_{t-1}), G_{\delta_1}(M \oplus A_t)\ \rangle$,
- $g(\cdot)$ is a test function for the variably hard function, $F(\cdot)$,
- $G_{\delta_i}(\cdot)$ are public hash functions $G_{\delta_i} : \{0,1\}^\kappa \to \{0,1\}^{\delta_i}$ in the random oracle model.

Figure 4.1: Header on day t for the degradation scheme with known schedule.

pression comes from the following two arguments. 1) For all valid f and j, A_{f-j+1} are different with probability at least $1 - \frac{L^2}{2^{\zeta\kappa}}$ (because, for $x > y$, when $A_x \in_r \{0,1\}^\kappa$, $A_x \neq A_y = W^{x-y}(A_x)$ with probability at least[7] $1 - O(2^{-\zeta\kappa})$, and there are $\leq L^2$ such pairs x, y). 2) For $(f,s) \neq (f', s')$, the values $M_{f,s} \oplus A_{f-j+1}$ and $M_{f',s'} \oplus A_{f'-j'+1}$ are equal with probability at most $2^{-\kappa}$ since $M_{f,s} \oplus M_{f',s'} = A_{f-j+1} \oplus A_{f'-j'+1}$ with probability $2^{-\kappa}$; there are at most $n_{sns}^2 L^4$ distinct pairs (f,s) and (f',s').

Let $1 - \Delta$ be the probability that all $M_{f,s} \oplus A_{f-j+1}$ differ one from another; from the argument above, $\Delta \leq \frac{L^2}{2^{\zeta\kappa}} + \frac{n_{sns}^2 L^4}{2^\kappa}$. Let λ be the probability of generating a successful call during $\mathcal{A}$'s execution, given that all $M_{f,s} \oplus A_{f-j+1}$ differ.

Suppose $\mathcal{A}$ makes a successful call given that all $M_{f,s} \oplus A_{f-j+1}$ differ. At the moment of the successful call, from the point of view of $\mathcal{A}$, all $G_{\delta_j}(M_{f,s} \oplus A_{f-j+1})$ are random variables drawn uniformly from $\{0,1\}^{\delta_j}$. Therefore, $\frac{\lambda}{n_{sns} L \tau} < 2^{-\zeta\kappa}$, since, otherwise, we could invert W as follows. Consider I_W, inverter for W: simulate $\mathcal{A}$ on input A_{t-i} and random values for the rest of the inputs; $\mathcal{A}$ makes some calls to the oracles; choose at random one of the calls to the oracles; guess values f, s; xor the input to the oracle with $M_{f,s}$ to obtain A_{f-j+1}; finally, compute $W^{-1}(A_{t-i}) = A_{t-i+1} = W^{t-f-i+j-2}(A_{f-j+1})$. Since W chooses at random among at most τ oracle calls made by $\mathcal{A}$, and among L, n_{sns} values for f and respectively s, I_W is successful with probability at least $\frac{\lambda}{n_{sns} L \tau}$. However any inverter has success probability of at most $2^{-\zeta\kappa}$, implying that $\frac{\lambda}{n_{sns} L \tau} < 2^{-\zeta\kappa}$.

Suppose $\mathcal{A}$ does not make a successful call, given that all $M_{f,s} \oplus A_{f-j+1}$ differ (which happens with probability $1 - \lambda$). In this case, all of $\mathcal{A}$'s input except A_{t-i} is distributed uniformly random from $\mathcal{A}$'s point of view. Furthermore, since $\mathcal{A}$ makes $\tau = O(2^{ce_i})$ operational effort, by a simulation argument as above, $\mathcal{A}$ can compute $F(v)$ with probability no better than stated in the definition of the hard function, i.e., at most $O(2^{-ce_i})$.

Concluding, we have that

$$\begin{aligned} \pi \leq\ & Pr[\text{not all } M_{f,s} \oplus A_{f-j+1} \text{ differ}] + \\ & + Pr[\text{all } M_{f,s} \oplus A_{f-j+1} \text{ differ}] \cdot (\lambda + (1-\lambda) \cdot 2^{-ce_i}) \\ \leq\ & \Delta + (1 - \Delta) \cdot (n_{sns} L \tau 2^{-\zeta\kappa} + 2^{-ce_i}) \\ \leq\ & \frac{L^2}{2^{\zeta\kappa}} + \frac{n_{sns}^2 L^4}{2^\kappa} + (n_{sns} L \tau 2^{-\zeta\kappa} + 2^{-ce_i}) \end{aligned}$$

We can choose κ such that $ce_i < \zeta\kappa/2$, yielding $2^{ce_i} \cdot 2^{-\zeta\kappa} < 2^{-\zeta\kappa/2} < 2^{-ce_i}$. L, the number of degradation levels, and n_{sns}, the number of session per day, are constants. Thus, we have that $\pi \leq O(2^{-\zeta\kappa}) + O(2^{-\kappa}) + O(2^{-ce_i}) = O(2^{-ce_i})$. □

4.2 Unknown degradation schedule

We now consider the more general online service setting in which the degradation scheme is unknown. That is, the center does not know when or if a user will be degraded or revoked, or when or if the user will be reinstated as a privileged user.

As an example of this setting consider a user who subscribes to an online service that bills the user periodically. If a bill payment is overdue, the service for that user is gradually degraded until either the user pays the bill and their service is subsequently reinstated (i.e. they rejoin class P), or the user does not pay the bill and their service continues to be degraded until the user is revoked and the service becomes unavailable to the user.

To model degradation in this subscription setting we again consider time as divided into sessions. In each session $\mathcal{C}$ may broadcast what we call a $Prefix$. A $Prefix$ enacts a change in hierarchy (i.e. a change in one or more of P, D or R). Regardless of whether or not a $Prefix$ is sent, $\mathcal{C}$ always sends a header, $Head$, at the beginning of each session (and immediately after any $Prefix$). The purpose of the header is the same as before, namely, to establish a new session key, SK, in such a way that each user must do the operational work to recover the session key that's specified by their class. The service content is then broadcast encrypted under the session key SK. The contents of $Prefix$ and $Head$ are made precise later in this section.

Because in this setting the degradation schedule is unpredictable, any degradation scheme needs the ability to degrade (or revoke) an arbitrary set of users. To accomplish this we leverage an efficient revocation scheme and the ideas from Section 4.1. A description of the scheme follows.

The initialization algorithm, $\mathsf{Params}(1^\kappa)$, establishes:

- A hard function $F(\cdot)$, and a corresponding test function $g(\cdot)$;
- A one-way permutation $W : \{0,1\}^\kappa \to \{0,1\}^\kappa$, such that any algorithm with time at most $2^{\zeta\kappa}$ inverts $W(\cdot)$ with probability less than $2^{-\zeta\kappa}$;
- $\delta_i = e_i - e_{i-1}$ (assuming $e_0 = 0$ and $e_{L+1} = \kappa$);
- A revocation scheme $\mathcal{S}$, such that, for any set $S \subset \mathcal{N}$ and message m, $\mathcal{S}$ can produce $\mathcal{E}^S(m)$ that can *only* be decrypted by users in S (see Section 2). $\mathcal{S}$ could be the SD [21] or LSD [14] revocation schemes, or, if maintaining current user state isn't a problem, then the stateful scheme of Appendix A can be used. Let $K_u^{\mathcal{S}}$ represent the private information allocated to user u by the scheme $\mathcal{S}$ (note that $K_u^{\mathcal{S}}$ can be a simple key or a set of keys as in [21, 14, 5], and our construction in Appendix A).
- The sequence $\{A_t\}_t$, $t \geq 1$, such that $A_t = W(A_{t+1})$; this series can be set-up by choosing a random value

[7] Since W is a one-way permutation, W^{x-y} is also a one way permutation against time $2^{\zeta\kappa}$. In particular, this means that the output of W^{x-y} is equal to the input with probability at most $O(2^{-\zeta\kappa})$.

$$Head^{unknown} = E_{K_{\bar{R}}}\left[\langle t, M, g(v), E_{K_P}(SK), E_{F(v)}(SK)\rangle\right]$$

- $M \in_r \{0,1\}^\kappa$,
- $v = \langle\ G_{\delta_{L+1}}(M \oplus A_{t-L}), G_{\delta_L}(M \oplus A_{t-L+1}), \ldots, G_{\delta_2}(M \oplus A_{t-1}), G_{\delta_1}(M \oplus A_t)\ \rangle$,
- $g(\cdot)$ is a test function for the variably hard function, $F(\cdot)$,
- $G_{\delta_i}(\cdot)$ are public hash functions $G_{\delta_i} : \{0,1\}^\kappa \to \{0,1\}^{\delta_i}$ in the random oracle model.

Figure 4.2: Header structure in session t for the degradation scheme with unknown schedule and unchanged user hierarchy ($h_{j-1} = h_j$).

$A_{t+\alpha}$, where α is a big integer (as mentioned in the first construction, it is easy to reduce α to L);

- Integer t that specifies which subset of $L+1$ values amongst $A_1, A_2, \ldots$ is currently used in the encryption; t is increasing over time (although the size of the subset, $L+1$, stays the same), specifically, t increases by 1 when the center $\mathcal{C}$ degrades users from $C_1, \ldots C_L$ even further (i.e. only after an instance of $Prefix_{lower}$ has been broadcast);
- A key K_P with an initial value of K_P^0 that we call the *privileged-user key*;
- A key $K_{\bar{R}}$ with an initial value of $K_{\bar{R}}^0$ that we call the *nonrevoked-user key*.

The key distribution, KeyGen, works as follows. When user u subscribes in session t, u obtains K_u^S as specified by $\mathcal{S}$, as well as the current values of K_P, $K_{\bar{R}}$, A_{t-1}. With these keys, u can decrypt the content in session t with a constant number of operations (i.e. without any degradation "penalty").

The header generation algorithm, Encrypt, can generate several possible header structures; the choice of the particular structure depends on whether or not the user hierarchy changes. For ease of exposition, we first consider the header when the hierarchy does not change (Figure 4.2). The header structure is very similar to that of Section 4.1.

To ensure the properties of a degradation scheme over all sessions, we preserve the following properties for current $t, K, K_{\bar{R}}$, and term them the invariant, $\mathcal{I}$:[8]

- Each privileged user $u \in P$ knows the current K_P, $K_{\bar{R}}$, as well as A_{t-1};
- Each degraded user $u \in C_i \subset D$, $1 \le i \le L$, knows A_{t-i}, the current $K_{\bar{R}}$, but not the current K_P;
- Any revoked user $u \in R$ does not know neither current $K_{\bar{R}}$ nor the current K_P;
- No user knows A_t for the current value of t.

For a header, $Head^{unknown}$, the decryption algorithm, KeyRecovery, works as follows. A privileged user $u \in P$ can simply decrypt SK using keys $K_{\bar{R}}$ and K_P. A degraded user $u \in C_i$ decrypts $g(v)$, computes the hint $y^{e_i} = \langle G_{\delta_{L+1}}(M \oplus A_{t-L}), \ldots, G_{\delta_{i+1}}(M \oplus A_{t-i})\rangle$, and finally $F(v)$ to decrypt SK. A revoked user will be unable to decrypt the header at all.

The harder case is when the user hierarchy changes. To describe this case, we specify first the allowed hierarchy transitions (i.e., the allowed changes in users hierarchy). To model our scenario of service subscription, we need the following hierarchy transitions:

Degrade: a set S of users from the privileged set P move to the set D of degraded users. Specifically, these users leave the class P of privileged set and join C_1, the level-one degradation class. The center will employ this transition when the users in S are late on payments, and the center wants to initiate gradual degradation of the service for the users in S.

Lower: further degradation of users in D, that is the class C_1 becomes C_2, the class C_2 becomes C_3, and so forth. In particular, the users from the class C_L become revoked, i.e, they leave the class D of degraded users and join the class R of revoked users. The center employs this transition periodically to continue the gradual degradation of service for users from D.

Revoke: a set S from $P \cup D$ is revoked. Specifically, each user $u \in S$ leaves its class and joins the class R of revoked users. The center employs this transition when it detects misbehavior by the users in S (e.g. piracy).

Raise: a set S of users is raised from a lower level to the class P of privileged users. The center will raise the level of users S when these users pay overdue bills and thus do not need to be degraded (or revoked) anymore.

Each transition is enforced by a broadcast of a corresponding header prefix $Prefix$, which is followed by the header $Head^{unknown}$ defined in Figure 4.2.[9] Figure 4.3 presents the prefix structures for each of the four transitions, as well as the resulting modifications to the hierarchies and state.

For each type of hierarchy transition, the table shows the actions of the center. The center performs these actions before it broadcasts the corresponding prefix. Each of the prefixes serve to update the set of keys each user knows in order to preserve the invariant $\mathcal{I}$. Note that, in fact, we can prepend several prefixes to the header $Head^{unknown}$.

Given the invariant $\mathcal{I}$, the security discussion is very similar to the one given in Section 4.1.1 since the structure of the header $Head^{unknown}$ is largely the same with $Head^{known}$ from the Section 4.1.1 (except that $Head^{unknown}$ is encrypted with the key $K_{\bar{R}}$). The correctness of the invariant $\mathcal{I}$ itself is easily verified – because all the class transitions rely on the encryption in the underlying revocation scheme $\mathcal{S}$.

In addition, collusion-resistance for the revoked users follows from the invariant and the fact that $Head^{unknown}$ is encrypted with the key $K_{\bar{R}}$.

5. SIMULATION

[8]Note that given this invariant it benefits a user to always be online and thus recover any updates of K_P and $K_{\bar{R}}$ that they can.

[9]Although it is not necessary to encrypt all five terms of the bracketed expression in the figure under $K_{\bar{R}}$, since doing so doesn't increase the cost of the scheme significantly we choose to encrypt them all the simplify the proof and exposition.

Transition	Center action	Prefix structure
Degrade	$P \leftarrow P \setminus S$ $C_1 \leftarrow C_1 \cup S$ $K_P \in_r \{0,1\}^\kappa$	$Prefix_{degrade} = \langle \mathcal{E}^P(K_P) \rangle$
Lower	$t \leftarrow t+1$ $R \leftarrow R \cup C_L$ $C_i \leftarrow C_{i-1}, i \in [2, L]$ $C_1 \leftarrow \emptyset$ $K_{\bar{R}} \in_r \{0,1\}^\kappa$	$Prefix_{lower} = \langle \mathcal{E}^{P \cup D}(K_{\bar{R}}), E_{K_P}(A_{t-1}) \rangle$
Revoke	$R \leftarrow R \cup S$ $C_i \leftarrow C_i \setminus S, 0 \le i \le L$ $K_P \in_r \{0,1\}^\kappa$ $K_{\bar{R}} \in_r \{0,1\}^\kappa$	$Prefix_{revoke} = \langle \mathcal{E}^P(K_P), \mathcal{E}^{P \cup D}(K_{\bar{R}}) \rangle$
Raise	$C_i \leftarrow C_i \setminus S, 1 \le i \le L+1$ $P \leftarrow P \cup S$	$Prefix_{raise} = \langle \mathcal{E}^S(K_P, K_{\bar{R}}, A_{t-1}) \rangle$

Figure 4.3: Prefixes in session t of the header for degradation schedule with unknown schedule in the case of a user hierarchy change. The four prefixes correspond to the four possible hierarchy changes: `degrade`, `lower`, `revoke`, and `raise`. Each of the prefixes can be followed by header $H^{unknown}$ to broadcast the session key itself.

In this section we present two simulations to demonstrate how our schemes can be used in practice. Our timings are based on the experiments of [8].

In the first simulation, we use a specific schedule for the degradation of users. In particular, we choose specific values for the degradation parameters (L and $\{e_i\}_i$) in the scheme of Section 4.2 that achieve graceful service degradation. In the second simulation, we analyze the broadcast communication overhead of our schemes in a representative instantiation of the Section 4.2 degradation scheme.

Degradation schedule. Recall that the degradation scheme of Section 4.2 can be naturally applied to broadcast-channel subscription services such as pay-TV. For our simulation we assume all users have their bill due on the same day and it is a 30 day bill cycle. In each of the first 7 days users who have not paid that cycle's bill are degraded. A user who does not pay their bill during any of those 7 days is revoked on the 8th day. For simplicity of exposition, we assume any user either pays on the first day or does not pay on days 1 through 7, and hence is revoked. We further assume that all the users who are revoked during one pay cycle pay their bill in between day 8 of that bill cycle and day 1 of the next, and thus are reinstated at the beginning of the next 30-day billing cycle.

The users are equipped with a settop box that decrypts the content and computes the variably hard function. The considered settop is a GCT-AllWell STB3036N (specific details can be found in [8]).

For a variably hard function we choose a memory-bound function, and, specifically, the `MBound` function described in [8] (adapted as described in section 2.1.1).

As discussed earlier, "degraded service" takes the form of a delay in a user's ability to access the content (decrypt a session). That is, before each session, a degraded user has to expend operations, thus consuming time, computing the hard function and decrypting the header. A session corresponds to a show (e.g., a TV show) and is approximated to be one hour long. Some sample delays are shown in the Table 1. Table 1 also shows the values of e_i that are necessary to achieve the stated delays. The exact times needed to compute the hard function are estimated using the experimental results of [8]. Specifically, [8] claims that with a value of $e_i = 15$, the settop needs roughly 42 seconds to compute the function `MBound` on an input.

Day	Delay	Frequency	e_i
1	42 secs	Once per session	15
2	1 min 24 secs	Once per session	16
3	2 mins 48 secs	Once per session	17
⋮	⋮	⋮	⋮
7	44 mins 48secs	Once per session	21

Table 1: An example of the delays imposed on the degraded users. A show is accessed by the degraded user with the specified delay. The day represents the day of the degradation period (which reflects the level of degradation).

Broadcast overhead. Next, we analyze the overhead in broadcast communication due to the additional headers that are needed by our degradation scheme.

We consider the same scenario as in the simulation above. Additionally, we assume that there are $n = 10^8$ subscribers. In any billing cycle, there are $d = 1\% \cdot n = 10^6$ subscribers that are late with their payment and these subscribers are degraded as a group in each billing cycle; eventually, all these users pay their bills and are reinstated, although for the simulation we assume they aren't reinstated until the next billing cycle.

Finally, we use LSD [14] as our revocation scheme $\mathcal{S}$ from 4.2. With the security parameter being $\kappa = 128$, in LSD scheme, a broadcast $\mathcal{E}^S(m)$ takes at most $4\kappa \cdot |m| \cdot |S| = 512 \cdot |m| \cdot |S|$ bits [14].

We compile the broadcast overheads in Table 2, and sort them by different sources of communication overhead; we also specify how frequently each overhead is incurred. We did not consider the effect of revoking the pirates due to lack of space and because it is not the focus of our paper. Note the prefixes can be broadcast during the night or other non-rush time.

6. CONCLUSION AND OPEN PROBLEMS

We have introduced the notion of degradation schemes for warning users of the impending end of online services. Our schemes rely on an extension of the moderately hard functions [1, 9] initially used for spam fighting, as well as the adaptation of existing revocation techniques. Our schemes are efficient in terms of communication overhead and user storage and have desirable collusion resistance. In addi-

Source	Frequency	Total #bits/unit of time
$Head^{unknown}$	1/session	$128 \cdot 5 = 640$/hour
$Prefix_{degrade}$	1/month	$512 \cdot d = 512$Mb/month
$Prefix_{raise}$	$1/(u \in D)$	$512 \cdot 3d = 1536$Mb/month
$Prefix_{lower}$	1/day	$512 \cdot \frac{23}{30} \cdot 2d = 785$Mb/day

Table 2: Upper bound on communication overhead according to its source. For each source, we list the frequency of the corresponding broadcast and the total number of bits in the overhead in a period of time.

tion, our general construction of Section 4.2 is designed to work with any revocation scheme. This has the advantages that the costs of our construction will be reduced with subsequent improvements in revocation and the approach is not restricted to the symmetric key setting (although we've focused on that here) but rather a public key revocation scheme can be used (see, for example, [7]).

To the best of our knowledge our work is the first to provide a warning mechanism that is bound to content retrieval and as in any new area such there is room for improvement. For example, we believe it is possible to extend an arbitrary hard function for use in a degradation scheme (as we demonstrated for the particular hard function of [8]) but a proof of this would be useful.

Additionally, future work might include a better parametrization of operational effort imposed on degraded users. At the moment, the difference in effort among degraded classes is exponential: a user in class C_i has to do an amount of work that is a factor of 2^{δ_i} larger than users in class C_{i-1}. Even if all $\delta_i = 1$, the work increases exponentially as the degraded classes become worse. To achieve a more controlled difference, we might, for example, consider using two (or more) variably hard functions in parallel such that a class C_i has to do work proportional to, say, $2^{e'_i} + 2^{e''_i}$.

7. REFERENCES

[1] M. Abadi, M. Burrows, M. Manasse and T. Wobber. Moderately hard, memory-bound functions. *Proceedings of the 10th Annual Network and Distributed Systems Security Symposium*, 2003.

[2] M. Abdalla, Y. Shavitt and A. Wool. Towards Making Broadcast Encryption Practical. *IEEE/ACM Transactions on Networking*, 8(4), pp. 443-454, August 2000.

[3] N. Attrapadung, K. Kobara, H. Imai. Broadcast Encryption with Short Keys and Transmissions. *Digital Rights Management Workshop 2003*, pp. 55–66.

[4] S. Berkovits. How to broadcast a secret. *Advances in Cryptology — Eurocrypt '91*, volume 547 of LNCS, pages 535–541. Springer-Verlag, 1991.

[5] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas. Multicast security: A taxonomy and efficient constructions. *IEEE INFOCOM*, 1999.

[6] Directory of Online Schools, http://www.directoryofonlineschools.com

[7] Y. Dodis and N. Fazio. Public key broadcast encryption for stateless receivers. *ACM CCS Workshop on Digital Rights Management 2002.*

[8] C. Dwork, A. Goldberg and M. Naor. On memory-bound functions for fighting spam. *Advances in Cryptology – Crypto 2003.*

[9] C. Dwork and M. Naor. Pricing via processing, or, combatting junk mail. *Advances in Cryptology –Crypto '92.*

[10] C. Dovrolis and D. Stiliadis. Relative differentiated services in the Internet: issues and mechanisms. *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems.*

[11] E. Engelking. Are you in favor of Microsoft's new XP licensing program? *Tech Republic*, May 4, 2001. http://techrepublic.com.com/5100-6270-1032935.html

[12] A. Fiat and M. Naor. Broadcast Encryption. *Advances in Cryptology – Crypto '93.*

[13] C. Gentry, Z. Ramzan. RSA Accumulator Based Broadcast Encryption. *7th Information Security Conference*, 2004.

[14] D. Halevy and A. Shamir. The LSD Broadcast Encryption Scheme. *Proceedings of the 22ns Annual International Cryptology Conference on Advances in Cryptology*, 2002, pp: 47-60.

[15] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security*, 1999.

[16] Keyhole, http://www.keyhole.com

[17] N. Kogan, Y. Shavitt and A. Wool. A Practical Revocation Scheme for Broadcast Encryption Using Smart Cards. *24th IEEE Symposium on Security and Privacy*, 2003.

[18] R. Kumar, S. Rajagopalan, A. Sahai. Coding Constructions for Blacklisting Problems without Computational Assumptions. *Proceedings of the 19th Annual International Cryptography Conference on Advances in Cryptology,* 1999.

[19] R. Motwani and P. Raghavan. Randomized Algorithms. *Cambridge University Press*, 2000.

[20] M. Naor and B. Pinkas. Efficient trace and revoke schemes. Financial Cryptogrpahy 2000.

[21] D. Naor, M. Naor, J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. *Advances in Cryptology – Crypto'01, Springer-Verlag LNCS 2139*, 2001, pp. 41-62.

[22] R. Rivest, A. Shamir and D. Wagner. Time-lock puzzles and timed-release crypto. *Massachusetts Institute of Technology Technical Report: TR-684*, 1996.

[23] A. Shamir. How to share a secret. *Communications of the ACM*, Volume 22 , Issue 11 (November 1979),pp. 612 - 613.

[24] D. Stinson. On Some Methods for Unconditionally Secure Key Distribution and Broadcast Encryption. *Designs, Codes and Cryptography*, 12 (1997), 215-243.

[25] U. C. Berkeley Extension Online. http://learn.berkeley.edu

[26] D.M. Wallner, E.J. Harder, R.C. Agee. RFC 2627 – Key management for multicast: Issues and architectures. Available at `http://www.rfc-archive.org`.

[27] C.K. Wong, M. Gouda, S.S. Lam. Secure group communications using key graphs. *Proceedings of*

ACM SIGCOMM, pp. 68–79, 1998.

APPENDIX

A. IMPROVED STATEFUL REVOCATION

In this section, we present a stateful revocation scheme that achieves $O(\log |R|)$ communication overhead per a revoked user, where R is the total number of revoked users. This is better than $O(\log n)$ communication overhead per a revoked user achieved by LKH-like stateful schemes ([26, 5, 27]).

The best known stateless revocation schemes achieve a communication blowup of $\Theta(R)$, where R is the total number of revoked users (except the scheme in [3], in which storage/computation cost is linear in the number of users, and they still need to broadcast $\Theta(R \log(n/R))$ bits to identify the revoked users).

With a stateful protocol, it is possible to do better when the users are revoked in rounds (e.g., when a batch pirates are found), which is usually the case in practice. Once the user is revoked in a round, it is considered revoked in all subsequent rounds (if a user needs to rejoin, the user is given a new identity that is transferred to the user via broadcast channel and is encrypted using keys received by the user at the initial subscription). Suppose r_t users are revoked in the round t. Then, a stateless protocol would have a communication blowup of $O(r_1 + r_2 + \dots r_t)$ in the round t. With a stateful protocol, it is concievable to achieve a communication blowup of only $O(r_t)$ in the round t.

Stateful protocols such as LKH were presented and improved, for example, in [26, 5, 27]. They all achieve a bound of $O(r_t \log n)$ communication blowup for the round t.

Here, we propose a new stateful revocation that achieves a communication blowup of $O(r_t \log(r_1 + r_2 + \cdots + r_t))$ for the round t. Note that $r_1 + r_2 + \cdots + r_t \leq n$, and, in fact, $r_1 + r_2 + \cdots + r_t$ is usually much less than n.

To achieve this bound we combine LKH scheme with subset-cover type revocation schemes ([21]). In a subset-cover revocation scheme, one has sets $S_i \subseteq \mathcal{N}$ such that for any set of revoked users R, the set $\mathcal{N} \setminus R$ can be represented as a union $\cup_{j=1}^{m} S_{i_j}$ for some i_j's. To each subset S_i, the scheme associates the key L_i; L_i is known by all users from S_i.

The subset-cover scheme needs also to satisfy the following two properties to be useful for our purposes:

- $\mathcal{N}$ is one of the sets S_i;
- For any S_i and $x \in S_i$, $S_i \setminus \{x\} = \cup_{j=1}^{c} S_{i_j}$, where S_{i_j} are disjoint and $c \leq \mathit{const}$. In other words, if we revoke a user, the subset cover changes only by a constant number of sets.

Note that these conditions are satisfied by the original SD scheme proposed by [21].

Now consider any LKH scheme which has $O(\log n)$ communication cost per a join/leave (revoke) operation, when there are n participants in the LKH scheme. Our scheme combines such an LKH scheme with subset-cover scheme to achieve the stated bound; specifically, we use the LKH scheme such that its participants ("users") are the sets S_i.

In the beginning, when nobody is yet revoked, $\mathcal{N}$ is the only participant in the LKH scheme. If a user x is revoked, then suppose $\mathcal{N} \setminus \{x\} = \cup_{j=1}^{c} S_{i_j}$. In this case, the participant $\mathcal{N}$ leaves the LKH scheme, and the participants $S_{i_1}, S_{i_2}, \dots S_{i_c}$ join the LKH scheme. Note that the center can communicate with the new participants by using their respective keys $L_{i_1}, L_{i_2}, \dots, L_{i_c}$. Communication cost is $O(c)$.

In general, suppose $S_{i_1}, \dots, S_{i_m}$ are participants of the LKH scheme (S_{i_j}'s are disjoint and $\cup_{j=1}^{m} S_{i_j} = \mathcal{N} \setminus R$, where R is the set of the users revoked so far). Let x be a user that needs to be revoked and suppose $x \in S_{i_j}$. Then we revoke the participant S_{i_j} from the LKH scheme. Also, if $S_{i_j} = S_{l_1} \cup \dots S_{l_c}$, then we join participants $S_{l_1}, \dots S_{l_c}$ to the LKH scheme. The communication cost is again $O(c)$.

After R revocations, there are $O(R)$ participants ("users") in the LKH scheme. Therefore, with earlier notations, the communication cost for round t is $O\left(r_t \log(r_1 + \cdots + r_t)\right)$.

The storage at user side is equal to the storage needed for the used subset-cover scheme plus the storage needed for the LKH scheme.

B. NOTATION

Notation	Definition	Section
$\mathcal{C}$	Center, distributor of the content	2
$\mathcal{N}$	The set of all users	2
P, C_0	The set of privileged users	2
D	The set of degraded users	2
R, C_{L+1}	The set of revoked users	2
$\mathcal{L}$	The set of all classes	2
L	Number of degraded classes	2
$C_1, \dots C_L$	Degraded classes, a partition of D	2
h_j	The hierarchy of users $h_j : \mathcal{N} \to \mathcal{L}$	2
κ	Security parameter	2
$F(v)$	A hard function	2.1
$g(v)$	A test function for $F(v)$	2.1
$Y^{(\ell)}$	A hint implying $\Omega(2^{c\ell})$ effort	2.1
c	A constant less than 1 characteristic to $F(v)$ and $g(v)$	2.1
e_i	Hardness parameter; implies 2^{ce_i} effort for class C_i	3
δ_i	$\delta_i = e_i - e_{i-1}$	4.1
Notation	Definition	Section
SK	A session key	2
T	Number of days in a trial period	4.1
n_{sns}	Number of sessions per day	4.1.1
W	A one-way function against time $2^{\zeta\kappa}$	4.1, 4.2
K_t	A series $\{K_t\}_t$ such that $K_t = W(K_{t+1}$	4.1
A_t	A series $\{A_t\}_t$ such that $A_t = W(A_{t+1}$	4.1, 4.2
t	Day number from the start of the protocol	4.1
t	An integer specifying which subset of the values $A_1, A_2, \dots$ is currently used in the encryption; t increases by 1 when $\mathcal{C}$ degrades users from $C_1, \dots C_L$ even further	4.2
G_{δ_j}	A hash function from $\{0,1\}^{\kappa}$ to $\{0,1\}^{\delta_j}$ in random oracle model	4.1, 4.2
M	A random value chosen uniformly at random from $\{0,1\}^{\kappa}$	4.1, 4.2
$\mathcal{S}$	A general revocation scheme, either stateless or stateful	4.2
$\mathcal{E}^{S}(\cdot)$	Encryption under scheme $\mathcal{S}$, targeted at a set S	2
K_P	A key known only by the priveledged users	4.2
$K_{\bar{R}}$	A key known only by the nonrevoked users (priveledged and degraded users)	4.2