Sliding-Window Self-Healing Key Distribution

Sara Miner More
University of California, San Diego
La Jolla, CA 92093-0114
sminer@cs.ucsd.edu

Michael Malkin
Stanford University
Stanford, CA 94305-9045
mikeym@stanford.edu

Jessica Staddon
Palo Alto Research Center
Palo Alto, CA 94304
staddon@parc.com

Dirk Balfanz
Palo Alto Research Center
Palo Alto, CA 94304
balfanz@parc.com

ABSTRACT

We propose a new method for distributing a common key to a dynamic group over an unreliable channel. In [15], an unconditionally secure "self-healing" protocol that solves this problem and has significant advantages over previous work in this area is presented. However, the protocol suffers from inconsistent robustness, high overhead and expensive maintenance costs. We propose a more practical self-healing protocol that attempts to address these three problems. First, we use a *sliding window* to make error recovery consistently robust. Second, we significantly reduce overhead. Finally, we give the group manager the ability to spread the cost of personal key distribution over multiple sessions, rather than having to distribute new personal keys to all users at the same time.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Algorithms

Keywords

Self-healing key distribution, sliding-window self-healing, multicast, broadcast encryption

1. INTRODUCTION

To enable secure group communication, it is common to have a group manager establish and periodically update a common key to be shared amongst the current group members (see, for example, [2, 17, 16, 3]). Since a group may be quite large, the group key is distributed via a broadcast channel in an encrypted form, so that only group members can decrypt it.

One possible attack on such a system is network disruption. If an attacker successfully shuts down the network channel used by a particular group member, the member may not receive key update messages. Even a short disruption can be damaging, since missing an update causes a member to be out of sync with the rest of the group, and can render him unable to decrypt group communication upon his return to the network. Depending on the system, it may be impossible for a user who misses only a single broadcast to recover without sending a message to the group manager. The manager could then either re-broadcast the update message, or set up a secure individual connection with the member himself, over which the latest group key is transmitted to the member directly. Re-broadcasting messages is expensive in terms of network resources, while requiring individual interaction for recovery from one lost update message places a burden on the group manager, particularly in very large groups. Furthermore, in some groups, requiring a member to initiate a conversation with the group manager may pose a security risk. For example, in a military application, particular group members may not wish to reveal their current locations by sending messages.

For both efficiency and security reasons, a non-interactive solution to this key distribution problem is desirable. We would like the affected group member to recover from lost update messages on his own – that is, *self-heal* – from the network disruption.

In [15], an unconditionally secure, non-interactive, reliable key distribution method for self-healing key distribution is presented as Construction 3. However, when considering a direct implementation of it, we note the following three areas for improvement:

- Robustness: At some stages of the protocol, if broadcast update messages are lost, the corresponding group key *cannot* be recovered, no matter how many other update messages are received. That is, the level of robustness is inconsistent.
- Overhead: For moderate system and security parameter values, update message length and personal key storage sizes become very large.

SSRS '03, October 31, 2003, Fairfax, Virginia, USA

- Personal key distribution: Periodically, new personal keys are distributed to all members through either *individual* TCP sessions, or off-line means, both of which can be quite inefficient in a large group.

In this paper, we modify techniques developed in [15] to construct a self-healing key distribution protocol with revocation which is more robust and requires less overhead. We implement self-healing in a *sliding-window* fashion, making the protocol uniformly robust: after an initial start-up period, the key contained in any lost update message can be recovered as long as two *sufficiently close* updates – one before it and one after it – are received. In addition, the size of the update message is significantly reduced. Finally, although our protocol does still require individual TCP sessions for the distribution of new personal keys, the individual distribution messages may be spread over the sessions, and users may be given personal keys for varying numbers of sessions.

One slight negative which results from these improvements, though, is that we reduce the amount of collusion resistance by one; that is, the protocol can only withstand coalitions of $t-1$ *active* group members. (Specifically, any coalition of t active group members is able to determine the personal key of another active user.)

1.1 Related Work

In addition to the work of Staddon et al. [15] mentioned above, Wong and Lam [18] and Perrig et al. [13] also deal with the problem of reliable group key distribution through non-interactive means. In [18], forward error correction is used to ensure that a group member who misses a broadcast is able, after sufficiently many subsequent broadcasts are received, to recover the missed group key. The private keys stored by the users form a hierarchical system (see, for example, [16, 17]). We note that when redundancy is used to achieve robustness in this way, neither revoked members nor recently joined members should receive sufficient redundancy to reconstruct information to which they are not entitled. This requirement may increase broadcast size.

In [13], hierarchical private key systems and redundant encoding of past group keys are also used to achieve reliable group key distribution. However, in [13], the key redundancy piggybacks on the actual content (i.e. the messages). In contrast, we allow for the possibility that the group manager is not the sole content distributor, so session key distribution must be separate from content.

Reliability is achieved through different means in [15]. Redundancy is used to encode both past and *future* group keys to ensure that a group member with broadcasts that "sandwich" a lost broadcast can recover the corresponding group key. Using this sandwiching recovery condition enables relatively small broadcasts (broadcast size is essentially independent of the size of the group) to be used even though the personal key system is flat rather than hierarchical. A flat key structure is conducive to traceability, because each personal key is known to only a single group member. We note here that, concurrent with and separate from our work, Liu and Neng [9] have built on the contributions of [15].

The main difference between our work and most others in the areas of multicast and broadcast encryption (e.g. [1, 6, 7, 8, 10, 11]) is that we address group key distribution over an unreliable channel. Additionally, we make use of several general techniques originally suggested in other works. In [12], Naor and Pinkas describe a secret-sharing-based revocation scheme in which the group manager broadcasts a user's secret in order to revoke him. This dynamic revocation technique is extended in [15], and we also use it here. Later in the paper, we sketch an extension to our protocol which allows self-enforcement. Originally suggested by Dwork, Lotspiech and Naor in [5], and also used in [12], the goal of self-enforcement is to deter users from sharing their private keys by tying keys to users' personal information, such as Social Security numbers.

2. NOTATION AND DEFINITIONS

We use essentially the same notation as described in [15]. We let $\mathcal{B}_j$ represent the key update broadcast during session j, and S_i represent user U_i's personal key. However, rather than having a fixed cluster of m sessions after which each user must be given new personal keys as in [15], we let m_i denote the session number after which user U_i needs to have his personal key updated. Note that two users U_i and $U_{i'}$ may have different values m_i and $m_{i'}$.

Also, we let $\mathcal{A}$ represent the group manager's list of group members (current active users). To denote the contents of $\mathcal{A}$ during a particular session j, we write $\mathcal{A}_j$, and we write $\mathcal{N}$ to denote the set of users who were never group members. We also need a way to represent the users who are revoked from the group. Recall that in Construction 3 in [15], revoked users are elements in the set $\mathcal{R}$, and their indices are placed in a set W during any broadcast in which they are revoked. This set W contains the indices of the personal keys which are sent in the broadcast. An active user who is to be revoked has, in his personal key, points on polynomials. One of these polynomials will be used in each future broadcast, so his index must be included in W for each future broadcast. In the sliding-window setting, however, the indices of users who are to be removed from the group do not need to be included in the set W of every future broadcast for the lifetime of the system – they only need to be included in broadcasts for which they already hold a corresponding point in their personal key. We call users whose points are being broadcast the current set of *actively revoked* users, and denote them by $\mathcal{R}$. Once they have been actively revoked in a sufficient number of consecutive broadcasts, users no longer have relevant personal keying information, and the effect is that they have the same key recovery ability from that point forward as would a user in $\mathcal{N}$, the set of never-active users. Thus, after a fixed number of sessions in which a user is actively revoked, he is moved from set $\mathcal{R}$ to set $\mathcal{N}$.

2.1 System Security Requirements

As in [15], one security parameter is t, a positive integer, which indicates how many group members the group manager can actively revoke in a given session. In that paper, t also indicates how resistant the system is to collusion: in order to find a session key to which they are not entitled, or to generate a new personal key, a collection of t or more group members must collaborate. Below, we use a more general name, c, for the collusion parameter. An additional security parameter, new in this work, is δ, which indicates how many consecutive missed update messages group members can tolerate (i.e., in some sense, the amount of network interruption from which users can self-heal), provided they have the corresponding personal keys. We make this precise in the definition below. Note that $H(\cdot)$ is the standard entropy function from information theory. (See [4], page 13.)

Definition 1. [Sliding-δ-Window Self-Healing Session Key Distribution with t-Revocation and c-Collusion Resistance] A sliding-δ-window session key distribution protocol with t-revocation capability and c-collusion-resistance consists of the following five algorithms:

- $Init$, a (possibly) randomized initialization algorithm, run by the group manager, which takes as input a set of security parameters, and outputs system parameters which are used as input to the remaining four algorithms.
- $AddUser$, a (possibly) randomized algorithm for adding a user to the set of active users $\mathcal{A}$, run by the group manager, which takes as input a user U_i and a session j, and outputs a personal key S_i to be used by U_i starting in session number j. In the process, user U_i is removed from either $\mathcal{N}$ or $\mathcal{R}$, and added to $\mathcal{A}$.
- $BroadcastGen$, a (possibly) randomized broadcast generation algorithm, run by the group manager, which takes as input a session $j \in \{1, \ldots, m\}$ and set of users $\mathcal{R}$ to be actively revoked, and outputs a key update message $\mathcal{B}_j$ to be broadcast during session j.
- $KeyRec$, a deterministic session key recovery algorithm, run by an individual user U_i who was in $\mathcal{A}$ during session j, which takes as input a broadcast message $\mathcal{B}_j$ and a personal key S_i, and outputs the session key for session j, which we call K_j.
- $SelfHealKeyRec$, a deterministic self-healing session key recovery algorithm, run by an individual user U_i, which takes as input personal key S_i, a session number j, and two broadcast messages $\mathcal{B}_b$ and $\mathcal{B}_a$, where $max\{j-\delta, 1\} \leq b < j < a \leq min\{j+\delta, m\}$ and U_i was in $\mathcal{A}$ during sessions b and a. It outputs the session key for session j, which we call K_j.

When the algorithm $Init$ is used to initialize the system parameters, the algorithm $AddUser$ is used to initialize each active user U_i in A with a personal key S_i, and algorithm $BroadcastGen$ is used to generate each broadcast $\mathcal{B}_j$, we require that the protocol meets the following three conditions, for each session $j \in \{1, \ldots, m\}$:

1. *Correctness:* For any $U_i \in \mathcal{A}_j$,
   (a) $KeyRec(\mathcal{B}_j, S_i)$ outputs K_j.
   (b) For sessions b and a such that $max\{j-\delta, 1\} \leq b < j < a \leq min\{j+\delta, m\}$ and $U_i \in \mathcal{A}_a$, $U_i \in \mathcal{A}_b$, $SelfHealKeyRec(S_i, j, \mathcal{B}_b, \mathcal{B}_a)$ outputs K_j.
2. *Security:*
   (a) $H(K_j) = |K_j|$.
   (b) $H(K_j|S_1, \ldots, S_n) = H(K_j)$.
   (c) For any i such that $U_i \notin \mathcal{A}_j$, $H(K_j|S_i, \mathcal{B}_1, \ldots, \mathcal{B}_m) = H(K_j)$.
3. *Collusion Resistance:* For any size-c or smaller coalition $C \subseteq (\mathcal{A} \cup \mathcal{R} \cup \mathcal{N})$ and any $U_i \in \mathcal{A}_j$ such that $U_i \notin C$,
   (a) $H(S_i) = |S_i|$.
   (b) $H(S_i|\{S_\ell\}_{U_\ell \in C}, \mathcal{B}_1, \ldots, \mathcal{B}_m) = H(S_i)$.
   (c) If $(C \cap \mathcal{A}_j) = \emptyset$, then $H(K_j|\{S_\ell\}_{U_\ell \in C}, \mathcal{B}_1, \ldots, \mathcal{B}_m) = H(K_j)$.

Given the above definition of a sliding-window self-healing key distribution protocol with revocation, we attempt to develop a protocol that meets it, with $c = t - 1$.

3. TECHNIQUES

This section highlights the techniques used in our new protocol, and the improvements of our protocol over those in [15]. Our full sliding-window self-healing session key distribution protocol is presented in Section 4.

Before discussing the techniques used in our main construction, we mention the following conventions. All operations will take place in the field F_q, where q is a prime number much larger than the expected total number of active and revoked users over the lifetime of the group communications. All random numbers are chosen uniformly from F_q, and all random polynomials are chosen uniformly from $F_q[x]$. N is a large element of F_q, and it may never be a user index. Finally, since the protocol presented here uses polynomial-based secret sharing [14], we will make use of polynomial interpolation in our constructions.

3.1 Masking Polynomials

In [15], when the group manager wants to privately transmit points on a t-degree polynomial to group members in a broadcast, he conceals the polynomial by adding to it another polynomial known as a *masking polynomial*. The masking polynomial is a bivariate polynomial of degree t in each variable, and is denoted $s(x, y)$. The univariate polynomial being masked is known as the *data polynomial*. In order for a group member to find his point on the data polynomial, he must first find a point on a specific univariate polynomial underlying the bivariate masking polynomial, and then subtract it from the data polynomial.

In this paper, we will use a similar technique, based on multiplication rather than addition. Specifically, the data polynomial will be multiplied by the masking polynomial, and the group member will divide away a point, rather than subtracting it. We make this more precise below.

Let $f(x)$ be a data polynomial, $s(x, y)$ be a random masking polynomial, and $s(i, i)$ be part of active user U_i's personal key. The group manager randomly chooses t points in F_q, denoted $w_1, w_2, ..., w_t$, such that none of the points is equal to any active user's index or the reserved constant value N, and broadcasts:

$$w_1, s(w_1, x), w_2, s(w_2, x), ..., w_t, s(w_t, x), f(x) \cdot s(N, x)$$

A group member U_i evaluates all of these polynomials at $x = i$. These t points plus the member's personal key give the member $t + 1$ points on the polynomial $s(x, i)$. These points are used to interpolate $s(x, i)$ and find $s(N, i)$. Finally, he evaluates $f(x) \cdot s(N, x)$ at $x = i$, and divides that by $s(N, i)$ to find $f(i)$. A revoked user w_i cannot perform such an interpolation, however, since he has only t distinct points on $s(x, i)$. Note that, for a user to be *permanently* revoked from the protocol, he must be revoked in each session for which he holds a corresponding point in his personal key. We opt for the multiplicative masking method here, rather than the additive one of [15], because it works with the technique for achieving smaller broadcasts described in Section 3.3.

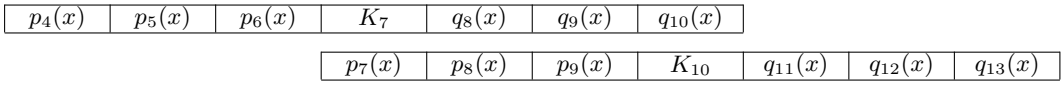

Figure 1: Information available in broadcasts at sessions 7 and 10 with $\delta = 3$.

3.2 Sliding-Window Self-Healing

Just as in Construction 3 in [15], each broadcast in this paper includes extra information which enables users to perform self-healing. As before, each key is represented as the sum of two polynomials, $p(x)$ and $q(x)$, so if an active user misses broadcast $\mathcal{B}_j$ but computes $p(i)$ and $q(i)$ from other broadcasts, he can calculate $K_j = p(i) + q(i)$.

However, in the constructions of [15], group communication is divided into clusters of m sessions. Update messages contain self-healing information for all other sessions in the cluster. Therefore, when a user is revoked, the group manager must actively revoke the user for the rest of the cluster. Furthermore, if an update message at very beginning or end of a cluster is lost, it cannot be recovered via self-healing.

In contrast, the protocol presented in this paper uses a *sliding-window* mechanism, where sessions are not grouped into fixed clusters. Instead, as time progresses, the group manager broadcasts self-healing information according to a sliding window that extends δ sessions in each direction from the current one. That is, in each session, the current session key is broadcast, along with self-healing information for the previous δ sessions and for the upcoming δ sessions. More precisely, in session j, the following values are broadcast: $p_l(x)$, for $l \in [j-\delta, j-\delta+1, ..., j-1]$, K_j, and $q_l(x)$, for $l \in [j+1, j+2, ..., j+\delta]$. (Note that these values are not sent in the clear; they are masked so that only active users can recover them.) In the example in Figure 1, we show the (unmasked) values present in two broadcasts. Because $p_8(x) + q_8(x) = K_8$ and $p_9(x) + q_9(x) = K_9$, one can see that the two broadcasts shown give enough information to find K_7, K_8, K_9, and K_{10}. The information provided in the broadcasts ensures that a user who is active over the entire lifetime of the protocol can recover all session keys as long as he never misses more than δ consecutive broadcasts. (In fact, the broadcasts will not allow a user to recover the entire polynomials mentioned above; they will only receive individual shares of each polynomial, to prevent collusion resistance.)

3.3 Smaller Broadcasts

Construction 3 in [15] uses a unique masking polynomial for each data polynomial sent in every broadcast. Since each broadcast includes m polynomials, information to interpolate m masking polynomials is sent in each broadcast, resulting in a broadcast size of $O(mt^2 + mt)$. In the sliding-window framework, broadcasts using this approach would have size $O(\delta t^2 + \delta t)$. However, we introduce a new technique which allows us to achieve $O(t^2 + \delta t)$ broadcast sizes.

In order to achieve shorter broadcasts, we will use only one masking polynomial $s_j(x, y)$ for all data polynomials in a single broadcast, instead of using a different masking polynomial for each data polynomial.

However, when $\delta \geq 1$, we cannot use the same polynomial $s_j(x, y)$ to mask the session key K_j (which is also data, but is not a data *polynomial*), for the following reason. Any coalition which includes an active user for a particular session j can recover session key K_j. If we broadcast $K_j \cdot s_j(N, x)$, then any coalition which includes an active member can learn the polynomial $s_j(N, x)$, and hence, can determine $s_j(N, i')$ for any user $U_{i'}$. After learning $s_j(N, i')$, the coalition can determine the personalized shares of user $U_{i'}$ sent in $\mathcal{B}_j$, e.g, $p_{j-1}(i')$. Specifically, consider the case where $U_{i'}$ is a member of the coalition who was a group member during session $j-2$, but actively revoked during session $j-1$. $U_{i'}$ already knows $q_{j-1}(i')$ from $\mathcal{B}_{j-2}$. Adding this value to $p_{j-1}(i')$ would allow the coalition to recover K_{j-1}, even if none of them was an active member during session $j-1$.

To avoid this type of attack, we use a different polynomial, called $r_j(x, y)$, to mask session key K_j. This prevents a curious group member U_i, who legitimately learns information about r_j in his recovery of K_j, from learning anything about the other (personalized) information to which only a different user $U_{i'}$ is entitled. A different pair of masking polynomials $(r_j(x, y), s_j(x, y))$ is used for every broadcast. Figure 2 illustrates the contents of broadcast $\mathcal{B}_j$.

Masked Shares	$p_{j-\delta}(x) \cdot s_j(N, x)$
	$\vdots$
	$p_{j-1}(x) \cdot s_j(N, x)$
Masked Key	$K_j \cdot r_j(N, x)$
Masked Shares	$q_{j+1}(x) \cdot s_j(N, x)$
	$\vdots$
	$q_{j+\delta}(x) \cdot s_j(N, x)$

Figure 2: Two masking polynomials per broadcast.

However, the particular implementation described in Figure 2 allows an attack on the system when $\delta \geq 2$, whereby a member who is active in time $j-1$ and revoked in time j can still find K_j. Specifically, in session $j-1$, if user U_i is active, he can remove the masking polynomial from $p_{j-2}(x) \cdot s_{j-1}(N, x)$ to reveal $p_{j-2}(i)$. In session j, suppose user U_i is revoked, and hence unable to interpolate the value of $s_j(N, i)$ from the points broadcast by the group manager. However, U_i already knows the value of $p_{j-2}(i)$, so he can divide this out of $p_{j-2}(i) \cdot s_j(N, i)$ to find $s_j(N, i)$.

To circumvent this attack, we modify the system so that knowledge of a data polynomial in one session does not completely reveal a masking polynomial used in a later session. This is done by splitting all of the data polynomials into two randomly-selected additive shares. New additive shares of a data polynomial are selected each session in which it is part of the broadcast. The two shares of a particular polynomial selected for broadcast $\mathcal{B}_j$ will be denoted with superscripts $^{(a_j)}$ and $^{(b_j)}$, such that $p_j^{(a_j)}(x) + p_j^{(b_j)}(x) = p_j(x)$ and

$q_j^{(a_j)}(x) + q_j^{(b_j)}(x) = q_j(x)$, for example. When two corresponding polynomial shares are masked in a broadcast, they will be masked with two different polynomials – two polynomials $s_j(x, y)$ denoted with the corresponding $^{(a_j)}$ and $^{(b_j)}$ superscripts. So, for example, $p_{j-1}^{(a_j)}(x)$ will be multiplicatively masked by polynomial $s_j^{(a_j)}(N, x)$. The contents of broadcast $\mathcal{B}_j$ are illustrated in Figure 3.

Masked Shares	$p_{j-\delta}^{(a_j)}(x) \cdot s_j^{(a_j)}(N,x)$, $p_{j-\delta}^{(b_j)}(x) \cdot s_j^{(b_j)}(N,x)$
	$\vdots$
	$p_{j-1}^{(a_j)}(x) \cdot s_j^{(a_j)}(N,x)$, $p_{j-1}^{(b_j)}(x) \cdot s_j^{(b_j)}(N,x)$
Masked Key	$K_j \cdot r_j(N,x)$
Masked Shares	$q_{j+1}^{(a_j)}(x) \cdot s_j^{(a_j)}(N,x)$, $q_{j+1}^{(b_j)}(x) \cdot s_j^{(b_j)}(N,x)$
	$\vdots$
	$q_{j+\delta}^{(a_j)}(x) \cdot s_j^{(a_j)}(N,x)$, $q_{j+\delta}^{(b_j)}(x) \cdot s_j^{(b_j)}(N,x)$

Figure 3: Splitting the masking polynomial $s_j(\cdot,\cdot)$ into shares.

Note that for a particular j in Figure 3, the polynomials $p_j(x)$ and $q_j(x)$ are constant over all update messages, while $p_j^{(a_j)}(x)$, $p_j^{(b_j)}(x)$, $q_j^{(a_j)}(x)$ and $q_j^{(b_j)}(x)$ are not. For example, $p_7^{(a_8)}(x)$ and $p_7^{(b_8)}(x)$ used during broadcast $\mathcal{B}_8$ will be different (respectively) from $p_7^{(a_9)}(x)$ and $p_7^{(b_9)}(x)$ used during broadcast $\mathcal{B}_9$ (except with negligible probability), but the sum of any two corresponding shares of the same session key will be equal. That is,

$$p_7^{(a_8)}(x) + p_7^{(b_8)}(x) = p_7(x) = p_7^{(a_9)}(x) + p_7^{(b_9)}(x).$$

Because the polynomials $p_j(x)$ and $q_j(x)$ are constant over all update messages, we must be careful that legitimate knowledge of a point on one of these polynomials does not help a user learn anything to which he is not entitled, should he later be revoked. To prevent this, we mask the additive shares of these polynomials multiplicatively, as shown in Section 3.1. For example, note that a user U_i first revoked during, say, session j_0, could have legitimately learned $p_{j_0-2}(i)$ from broadcast $\mathcal{B}_{j_0-1}$. Had we used additive masking instead of multiplicative masking, this user could attack the system after reading the public broadcast $\mathcal{B}_{j_0}$ as follows. First, he could evaluate the additively masked shares of $p_{j_0-2}(x)$ in $\mathcal{B}_{j_0}$ at $x = i$, giving $p_{j_0-2}^{(a_{j_0})}(i) + s_{j_0}^{(a_{j_0})}(N,i)$ and $p_{j_0-2}^{(b_{j_0})}(i) + s_{j_0}^{(b_{j_0})}(N,i)$. Summing these values and subtracting $p_{j_0-2}(i)$, he would learn $s_{j_0}^{(a_{j_0})}(N,i) + s_{j_0}^{(b_{j_0})}(N,i)$, since $p_{j_0-2}^{(a_{j_0})}(i) + p_{j_0-2}^{(b_{j_0})}(i) = p_{j_0-2}(i)$. He could then evaluate the polynomials $q_{j_0+\delta}^{(a_{j_0})}(x) + s_{j_0}^{(a_{j_0})}(N,x)$ and $q_{j_0+\delta}^{(b_{j_0})}(x) + s_{j_0}^{(b_{j_0})}(N,x)$ in $\mathcal{B}_{j_0}$ at $x = i$, and sum them, giving $q_{j_0+\delta}^{(a_{j_0})}(i) + s_{j_0}^{(a_{j_0})}(N,i) + q_{j_0+\delta}^{(b_{j_0})}(i) + s_{j_0}^{(b_{j_0})}(N,i)$. Subtracting off $s_{j_0}^{(a_{j_0})}(N,i) + s_{j_0}^{(b_{j_0})}(N,i)$, he would learn $q_{j_0+\delta}^{(a_{j_0})}(i) + q_{j_0+\delta}^{(b_{j_0})}(i) = q_{j_0+\delta}(i)$. However, this user was revoked during session j_0, and hence should not learn $q_{j_0+\delta}(i)$. Because the masking is done in a multiplicative way, however, it seems that knowledge of one point on a data polynomial does not reveal useful information about the sum of two masking polynomials.[1]

4. OUR PROTOCOL

In this section, we present our sliding-window self-healing session key distribution protocol, based on the techniques described in the previous section. The security and efficiency properties of the protocol are discussed in Section 5.1.

Construction 1. A Sliding-δ-Window Self-Healing, Session Key Distribution Protocol with t-Revocation Capability.

Algorithm Init: To prepare the system, the group manager randomly chooses q to be a prime number larger than the total number of different group members expected over the *lifetime of the system*, and randomly chooses N to be a non-zero element of F_q. He initializes the sets $\mathcal{A}$ and $\mathcal{R}$ to be empty. For each session $j \in \{2, \ldots, \delta + 1\}$, the group manager randomly selects a key K_j and degree-t masking polynomial $p_j(x)$, and sets $q_j(x) = K_j - p_j(x)$. Finally, for session $j \in \{1, \ldots, \delta + 2\}$, he randomly chooses masking polynomials $r_j(x, y)$, $s_j^{(a_j)}(x, y)$ and $s_j^{(b_j)}(x, y)$ of degree t in both variables.

Algorithm AddUser: To add a new user U to the group starting in session j until some session m, the group manager randomly selects an index i from F_q such that $i \notin (\mathcal{A} \cup \mathcal{R} \cup \{N\})$, and sets $m_i = m$. U is removed from the set where it currently appears($\mathcal{N}$ or $\mathcal{R}$), and his new index i is added to the set of active users $\mathcal{A}$.[2]

Next, for any session $j' \in \{j, j+1, \ldots, m_i\}$ for which he has not already done so, he generates new personal key polynomials. Specifically, for each such j', he randomly chooses masking polynomials $r_{j'}(x, y)$, $s_{j'}^{(a_{j'})}(x, y)$ and $s_{j'}^{(b_{j'})}(x, y)$ of degree t in both variables.

Then the new user, now referred to as U_i, will be given the values i, δ, N, and his personal key S_i will contain:

$$r_j(i,i),\ r_{j+1}(i,i), \ldots,\ r_{m_i}(i,i),$$
$$s_j^{(a_j)}(i,i),\ s_{j+1}^{(a_{j+1})}(i,i), \ldots,\ s_{m_i}^{(a_{m_i})}(i,i) \text{ and}$$
$$s_j^{(b_j)}(i,i),\ s_{j+1}^{(b_{j+1})}(i,i), \ldots,\ s_{m_i}^{(b_{m_i})}(i,i).$$

Algorithm BroadcastGen: If an active user is to be revoked starting in session j, the group manager removes his index from $\mathcal{A}$ and places it into $\mathcal{R}$. (This can be done repeatedly while the current number of actively revoked users (i.e., $|\mathcal{R}|$) is strictly less than t.) To revoke user U_i starting in session j, the group manager removes U_i from $\mathcal{A}$ and adds it to $\mathcal{R}$ prior to generating broadcast $\mathcal{B}_j$. (The index of user U_i will be removed from $\mathcal{R}$ and added to $\mathcal{N}$ just after broadcast $\mathcal{B}_{m_i}$ is

[1]Note that this technique does reveal some information about the masking polynomials. For example, anyone can recover the ratio of $p_{j-1}^{(a_j)}(x)$ and $p_{j-2}^{(a_j)}(x)$ from broadcast $\mathcal{B}_j$. As such, we are not immediately able to claim unconditional security for our protocol. See Section 5.1 for intuition about its security, however.

[2]If the user was most recently in $\mathcal{R}$, he must be given a new index value before being added to $\mathcal{A}$. Furthermore, the old index value must remain in $\mathcal{R}$ until the end of the active revocation period.

generated.) Then, the group manager randomly generates the set $\mathcal{W} = \{w_1, w_2, ..., w_t\}$ such that $\mathcal{R} \cap \mathcal{W} = \mathcal{R}$ and $\mathcal{A} \cap \mathcal{W} = \emptyset$ and the reserved constant $N \notin \mathcal{W}$.

To prepare the rest of broadcast $\mathcal{B}_j$, the group manager will use the previously-chosen personal key polynomials for the δ sessions which follow session j. These are: $r_\ell(x, y)$, $s_\ell^{(a_\ell)}(x, y)$ and $s_\ell^{(b_\ell)}(x, y)$ for $\ell \in \{j+1, \ldots j+\delta+1\}$.

In addition, he will use the polynomials p_ℓ which are p-shares for the previous δ sessions ($\ell = j-\delta, \ldots, j=1$), and the polynomials q_ℓ which are q-shares for the following δ sessions ($\ell = j+1, \ldots, j+\delta$), if those sessions exist. (There is no need for p shares of sessions prior to 1, or q-shares for sessions following the final one). He also uses the value of the current session key K_j. For each of the p_ℓ and q_ℓ polynomials mentioned above, he randomly selects an $^{(a_j)}$ share polynomial and the corresponding $^{(b_j)}$ share polynomial, so that $p_\ell^{(a_j)}(x) + p_\ell^{(b_j)}(x) = p_\ell(x)$, and $q_\ell^{(a_j)}(x) + q_\ell^{(b_j)}(x) = q_\ell(x)$. He then sends the broadcast $\mathcal{B}_j$ described in Figure 4.

Session number
j

Indices and polynomials
$w_1, r_j(w_1, x),\ s_j^{(a_j)}(w_1, x), s_j^{(b_j)}(w_1, x)$
$\vdots$
$w_t, r_j(w_t, x),\ s_j^{(a_j)}(w_t, x), s_j^{(b_j)}(w_t, x)$

Masked p polynomial shares
$p_{j-\delta}^{(a_j)}(x) \cdot s_j^{(a_j)}(N, x),\ p_{j-\delta}^{(b_j)}(x) \cdot s_j^{(b_j)}(N, x)$
$\vdots$
$p_{j-1}^{(a_j)}(x) \cdot s_j^{(a_j)}(N, x),\ p_{j-1}^{(b_j)}(x) \cdot s_j^{(b_j)}(N, x)$

Masked session key
$K_j \cdot r_j(N, x)$

Masked q polynomial shares
$q_{j+1}^{(a_j)}(x) \cdot s_j^{(a_j)}(N, x),\ q_{j+1}^{(b_j)}(x) \cdot s_j^{(b_j)}(N, x)$
$\vdots$
$q_{j+\delta}^{(a_j)}(x) \cdot s_j^{(a_j)}(N, x),\ q_{j+\delta}^{(b_j)}(x) \cdot s_j^{(b_j)}(N, x)$

Figure 4: The contents of broadcast $\mathcal{B}_j$.

Next, the group manager will prepare for session $j+1$. For any active user U_i for which the current session $j = m_i$, the group manager must individually send him new personal keys. For such a user, he runs algorithm AddUser, giving that individual the same index i already assigned to him; this will update that user's m_i value as well.[3] The group manager then generates a new session key and polynomial shares of the session key for session $j+\delta+1$, by randomly selecting a key $K_{j+\delta+1}$ and degree-t masking polynomial $p_{j+\delta+1}(x)$, and sets $q_{j+\delta+1}(x) = K_{j+\delta+1} - p_{j+2\delta+1}(x)$. Finally, if for any actively revoked user U_i, the value of m_i equals j, the group manager moves U_i from $\mathcal{R}$ to $\mathcal{N}$.

[3]In fact, this could be done ahead of time, before session m_i is reached.

Algorithm KeyRec: From broadcast $\mathcal{B}_j$, active member U_i with $m_i \geq j$ learns indices w_ℓ and polynomials $r_j(w_\ell, x)$ for $\ell = 1, \ldots, t$. He evaluates each of these polynomials at $x = i$, obtaining t points of the form $(w_\ell, r_j(w_\ell, i))$. Using these points and point $(i, r_j(i, i))$ from his personal key, he interpolates to get the polynomial $r_j(x, i)$. Evaluating this at $x = N$, he obtains $r_j(N, i)$. He then evaluates $K_j \cdot r_j(N, x)$ (obtained from broadcast $\mathcal{B}_j$) at $x = i$, and divides out $r_j(N, i)$, leaving the session key K_j. He can then delete $r_j(i, i)$ from his personal key.

Algorithm SelfHealKeyRec: From broadcasts $\mathcal{B}_b$ and $\mathcal{B}_a$, such that $max\{j-\delta, 1\} \leq b < j < a \leq min\{j+\delta, m\}$, user U_i with $m_i \geq a$ who was active in sessions b and a can recover session key K_j as follows.

From broadcast $\mathcal{B}_b$, U_i learns indices w_ℓ and polynomials $s_b^{(a_b)}(w_\ell, x)$ and $s_b^{(b_b)}(w_\ell, x)$, for $\ell = 1, \ldots, t$. He evaluates each of these polynomials at $x = i$, obtaining t points of the form $(w_\ell, s_b^{(a_b)}(w_\ell, i))$ and t of the form $(w_\ell, s_b^{(b_b)}(w_\ell, i))$. Using these values and points $(i, s_b^{(a_b)}(i, i))$ and $(i, s_b^{(b_b)}(i, i))$ from his personal key, he interpolates to get the polynomials $s_b^{(a_b)}(x, i)$ and $s_b^{(b_b)}(x, i)$. Evaluating these at $x = N$, he obtains $s_b^{(a_b)}(N, i)$ and $s_b^{(b_b)}(N, i)$. Then, he evaluates the polynomials $q_j^{(a_b)}(x) \cdot s_b^{(a_b)}(N, x)$ and $q_j^{(b_b)}(x) \cdot s_b^{(b_b)}(N, x)$ at $x = i$ and divides out the values $s_b^{(a_b)}(N, i)$ and $s_b^{(b_b)}(N, i)$, respectively, leaving him with shares $q_j^{(a_b)}(i)$ and $q_j^{(b_b)}(i)$. He then adds these together to give $q_j(i)$. He follows the analogous procedure to learn share $p_j(i)$ from broadcast $\mathcal{B}_a$, and adds the two shares to give session key K_j.

5. ANALYSIS OF OUR PROTOCOL

In this section, we discuss the security properties that must hold for self-healing key distribution as defined in Section 2.1, and provide evidence that they are satisfied by our protocol. For many properties, the evidence takes the form of a rigorous argument. However, we only provide intuition for the security of our technique for multiplicatively masking additive shares (see Section 3.3). A rigorous proof of the security of this technique is the focus of ongoing work.

5.1 Security Discussion

We start by proving the following technical lemma.

LEMMA 1. *Given a bivariate polynomial $s(x, y)$, with coefficients $a_{i,j}$, for $i, j \in \{0, \ldots, t\}$, uniformly selected at random from F_q such that $s(x, y) = \Sigma_{i,j=0}^{t} a_{i,j} x^i y^j$, the univariate polynomial $s(x, x)$ has coefficients which are also uniformly distributed at random over F_q.*

PROOF. Let the univariate polynomial $s(x, x)$ be called $s'(x)$, which we can express as $\Sigma_{k=0}^{2t} a'_k x^k$, where the a'_k's are the coefficients. Note that each a'_k is the sum of a subset of the coefficients $\{a_{i,j}\}$. Specifically, for all $k \in \{0, \ldots 2t\}$ and all $i, j \in \{0, \ldots, t\}$, we have $a'_k = \Sigma_{i,j:i+j=k}\ a_{i,j}$. Thus, the coefficients a'_k are the sums of disjoint subsets of the $a_{i,j}$'s, and hence are not correlated with one another. Specifically, since they are the sum modulo q of random elements of F_q, the coefficients of the univariate polynomial are distributed randomly over F_q as well. □

We now discuss the requirements of Definition 1 in turn. Our construction has sliding-window parameter δ, t-revocation capability, and $(t-1)$-collusion-resistance.

CORRECTNESS: For any $U_i \in \mathcal{A}_j$, both items in the correctness property follow directly from the construction, for any active user U_i who holds the appropriate points in his personal key. That is, the correctness property holds for any user U_i whose $m_i \geq j$.

SECURITY: The desired properties are written below as equations involving the entropy function. Each equation is followed by an argument that our protocol satisfies it.

1. $H(K_j) = |K_j|$: Since the session keys are selected independently at random from F_q, each of q values is equally likely. Thus, $H(K_j) = |K_j|$.

2. $H(K_j|S_1, \ldots, S_n) = H(K_j)$: We note that, for each j, the polynomials $s_j^{(a_j)}(x, y)$, $s_j^{(b_j)}(x, y)$, and $r_j(x, y)$ from which secret key points are taken are selected randomly and independent from the selection of any K_j, $p_j(x)$, or $q_j(x)$. Thus, the secret keys alone do not determine K_j.

3. For any i such that $U_i \notin \mathcal{A}_j$, $H(K_j|S_i, \mathcal{B}_1, \ldots, \mathcal{B}_m) = H(K_j)$: This statement follows directly from Condition 3 below.

COLLUSION RESISTANCE: For any coalition C of size strictly smaller than t, such that $C \subseteq (\mathcal{A} \cup \mathcal{R} \cup \mathcal{N})$, and any $U_i \in \mathcal{A}_j$ such that $U_i \notin C$, we want the following entropy equations to hold. Below each equation, we argue that our protocol satisfies it.

1. $H(S_i) = |S_i|$: An active user U_i's personal key consists of points on polynomials randomly selected over $F_q[x]$. Because the polynomials are selected from the uniform distribution over $F_q[x]$, the value of any point on that polynomial is equally likely to be any point in F_q and hence $H(S_i) = |S_i|$.

2. $H(S_i|\{S_\ell\}_{U_\ell \in C}, \mathcal{B}_1, \ldots, \mathcal{B}_m) = H(S_i)$:

   In order to learn information about the personal key of an active member who is not part of the coalition, the coalition must learn a point on a polynomial $r_j(x, x)$, $s_j^{(a_j)}(x, x)$, or $s_j^{(b_j)}(x, x)$ for some session j. From Lemma 1, each of these is a degree-$2t$ polynomial whose distribution is uniform over $F_q[x]$. Thus, the value of any point on them is equally likely to be any element of F_q. The coalition will of course know their own personal key points on each of the particular polynomials $r_j(x, x)$, $s_j^{(a_j)}(x, x)$, and $s_j^{(b_j)}(x, x)$. Because the coalition size is at most $t - 1$, though, this is at most $t - 1$ points each of the form $r_j(\ell, \ell)$, $s_j^{(a_j)}(\ell, \ell)$, and $s_j^{(b_j)}(\ell, \ell)$.

   From the portion of broadcast $\mathcal{B}_j$ which contains the indices and polynomials from set W, the coalition can learn at most t additional points on each of the three degree-$2t$ polynomials by evaluating each of the t univariate polynomials of the form $r_j(W, x)$, $s_j^{(a_j)}(W, x)$ and $s_j^{(b_j)}(W, x)$ at the respective W values. (Of course, none of these new points corresponds to an *active* user.)

   The portion of the broadcast containing the masked session key is useful for learning exactly one additional point of the form $r_j(\ell, \ell)$ (if the coalition includes an active member who can determine K_j).[4] The key K_j can be subtracted away to learn the $r_j(N, x)$ polynomial used to mask it, but this can only provide a point of the form $r_j(N, N)$, (Note that N is never a user point.) Furthermore, the rest of the broadcast is not useful for learning new points $s_j^{(a_j)}(\ell, \ell)$, or $s_j^{(b_j)}(\ell, \ell)$, since the $p_{j'}(x)$ polynomials which are multiplied by the s masking polynomials are split into randomly selected additive shares, causing the product to be randomly distributed over F_q.

   Because that part of the broadcast contains polynomials $p_{j'}^{(a_j)}(x) \cdot s_j^{(a_j)}(N, x)$ and $p_{j'}^{(b_j)}(x) \cdot s_j^{(b_j)}(N, x)$ and $q_{j'}^{(a_j)}(x) \cdot s_j^{(a_j)}(N, x)$ and $q_{j'}^{(b_j)}(x) \cdot s_j^{(b_j)}(N, x)$, the coalition might hope to determine a point on $p_{j'}(x)$ or $q_{j'}(x)$ to divide out from one of the polynomials, giving a point on $s_j^{(a_j)}(N, x)$ or $s_j^{(b_j)}(N, x)$. However, the only point on $s_j^{(a_j)}(N, x)$ which is also on $s_j^{(a_j)}(x, x)$ is $s_j^{(a_j)}(N, N)$, but N is never used as a user index. Furthermore, the only points the coalition knows on $p_{j'}(x)$ and $q_{j'}(x)$ are those points $p_{j'}(\ell)$ and $q_{j'}(\ell)$, where $U_\ell \in B$. There are at most t of each of these points, so the coalition could not determine any other points on $p_{j'}(x)$, $q_{j'}(x)$ or $s_j^{(a_j)}(N, x)$ which might help them determine a point on $s_j^{(a_j)}(x, x)$. The same argument can be made for the $s_j^{(b_j)}(x, x)$ case. Therefore, this is a maximum number of $2t$ points known to the coalition on each random degree-$2t$ polynomial, and so it seems that no additional points can be determined.

3. If $(C \cap \mathcal{A}_j) = \emptyset$, then $H(K_j|\{S_\ell\}_{U_\ell \in C}, \mathcal{B}_1, \ldots, \mathcal{B}_m) = H(K_j)$:

   Finally, if no coalition member is active during session j (i.e., $\mathcal{A} \cap B = \emptyset$), then the coalition will only have at most $t - 1$ distinct points of the form $r_j(w_i, x)$, so the value of K_j is not determined, even given $K_j \cdot r_j(N, x)$.

   Earlier, we saw that different users' shares $q_j(i)$ (obtained in a broadcast prior to session j) and $p_j(i')$ (obtained in a broadcast after session j) of K_j are incompatible, and the group manager will not allow the same user to leave the group before session j and return later using the same index.

5.2 Efficiency

We now compare broadcast sizes in our protocol with broadcast sizes in Construction 3 of [15]. Each broadcast in Construction 3 had size $O((t^2m + tm)k)$, where k represents the size of the session key, and m represents the number of sessions in a cluster. Our construction, however, achieves a broadcast size of $O((t^2 + t\delta)k)$, where our sliding-window size δ is generally much smaller than the parameter m in [15]. For direct comparison, if δ is set to m, our broadcast size is still smaller by a factor of m in the t^2 term.

Our construction also requires less personal key storage per user than Construction 3 in [15], where a group member must store a personal key of size $O(m^2k)$. In our construction, the user stores a key of size $O(m_ik)$, where m_i represents his "personal" cluster size. If we set each m_i to be the

[4]This is the cause of the value of the collusion-resistance parameter dropping from t in [15] to $t-1$ in our construction.

m used in [15] for the purposes of direct comparison, this is an improvement of a factor of m. In practice, the flexibility of our m_i values allows group managers to trade off the amount of required personal key storage and the frequency of individual key update messages as applications warrant.

6. EXTENSIONS

This section discusses several variants and an enhancement one could make to the session key distribution protocol we described in Section 4.

6.1 Sliding-Window Variants

The protocol presented so far uses a sliding-window that extends some constant δ_0 sessions into both the past and future (that is, our parameter δ is set to δ_0). However, other configurations are possible. The sliding-window can have arbitrary length both forward and backward, and these lengths may change over time. One simple variant would be to modify δ so that alternate packets have different window lengths. For example, for *every other* broadcast in the system, we could lengthen the sliding-window by 1 in each direction (for these packets, $\delta = \delta_0 + 1$), while, for all remaining packets, the size of the window is reduced to size 1 ($\delta = 0$). This significantly reduces the length of the broadcasts and, with high probability, will not reduce the self-healing ability of group members. (Members would be most vulnerable to noise that caused them to miss every other broadcast.) It does, however, add extra latency to self-healing, since members sometimes must wait an extra session to accumulate enough information to perform self-healing.

Another interesting variant is to use a randomized sliding-window in which not necessarily all sessions in the window are represented – that is, a broadcast might not include shares of some session in its window. For example, the decision to include a share of a particular session key might be made according to a Gaussian distribution. In that case, care must to be taken to ensure that the probability of self-healing was high according to the appropriate noise model.

6.2 Self-Enforcement

Self-enforcement is an idea first presented in [5], where a group member's personal key is linked to some personal information belonging to him, so that anyone who knows the member's key can access his personal information. The information in question could be a Social Security number, for example, or any other information the member would want to keep secret. This link between private keys and personal information is intended to serve as a deterrent to key sharing: many members will take extra care to keep their personal keys secret so that their personal information remains secret as well. On the other hand, in a system without self-enforcement, any group member can give his private key to anyone he chooses. Self-enforcement is meant to discourage this sort of behavior.[5]

Furthermore, because users may be revoked in later sessions, dishonest members may attempt to assemble a coalition of users who will work together to try to learn some secret of the system. If sufficiently motivated, say for significant financial gain, these members may be willing to sacrifice their personal information on a limited scale. However, if self-enforcement is incorporated into our system, even those willing to leak personal information to a few colluders might be affected by it in the following way. If the coalition of dishonest members is hoping for widespread distribution of a key to non-group members (e.g., they are planning to sell it on the Internet), it is unlikely that any of them will be willing to leak their information on such a large scale. This is where our security parameter t comes into play. In order to produce a key that will not leak any personal information about coalition members, the size of the coalition must be larger than $t - 1$.

Clearly, then, the inclusion of a self-enforcement mechanism would make the system described in Section 4 more robust. We achieve computationally-secure self-enforcement using the following mechanism, which is modeled on one presented in [12]. When a user wishes to be added to the group, he sends the group manager his personal information. (This protocol therefore requires a trusted group manager.) The group manager, after verifying the personal data, gives the new member a *randomly selected* index i. For user U_i, the personal information is known as $\mathcal{S}_i$, and is stored in a public database, encrypted via a symmetric cipher using $\mathcal{F}(i)$ as the key, where $\mathcal{F}(x)$ is a random t-degree polynomial. During session j, the group manager adds the following to the broadcasts:

$$\mathcal{F}^{(a)}(x) \cdot s_j^{(a)}(N, x), \mathcal{F}^{(b)}(x) \cdot s_j^{(b)}(N, x)$$

Here $\mathcal{F}^{(a)}(x)$ and $\mathcal{F}^{(b)}(x)$ are chosen randomly with each broadcast but $\mathcal{F}^{(a)}(x) + \mathcal{F}^{(b)}(x) = \mathcal{F}(x)$ is kept constant, similar to Section 3.3. Therefore, anyone with member U_i's personal key value $s_j(i, i)$ can discover $\mathcal{F}(i)$ and decrypt $\mathcal{S}_i$ in the public database. On the other hand, to someone without knowledge of the personal key value $s_j(i, i)$, the value $\mathcal{F}_i$ is still randomly distributed over F_q.

To speed up searching the database, each entry is indexed by $H(\mathcal{F}(i))$, where $H(x)$ is a public, randomly selected hash function with partial preimage resistance. Partial preimage resistance requires that, given an output y, it is computationally infeasible to find any part of a preimage x' such that $H(x') = y$. Thus, a third party with knowledge of $\mathcal{F}(i)$ can efficiently find user U_i's secret information in the database, but $\mathcal{S}_i$ is not revealed to everyone.

7. SUMMARY AND OPEN PROBLEMS

In this paper, we propose a practical self-healing session key distribution scheme which is a modified version of the protocols in [15]. Broadcast and personal key sizes in the protocol presented in that paper were rather large, however, and some practical issues were unresolved. In particular, the self-healing property did not stretch across clusters of sessions. We introduce two new techniques to overcome these shortcomings:

- **A sliding window** extends the self-healing property to apply δ sessions into the past and the future, independent of any session cluster size. Furthermore, different cluster sizes may be selected for different users, offering flexibility.

[5]Of course, self-enforcement does not discourage members from giving away *session* keys directly. However, this type of attack would require very frequent communication (as often as once per session) to allow the information to be decrypted in a timely manner. We do not address this threat here.

- **Reusing masking polynomials** reduces broadcast size and personal key storage significantly.

We provide evidence suggesting that our protocol achieves unconditional security. (As future work, we would like to rigorously prove its security.) We also discuss possible variants of our self-healing key distribution protocol and propose an adaptation of self-enforcement to it.

It would be interesting to consider other personal key distribution methods, such as one based on our technique to distribute individual shares to users via broadcast. Specifically, we would like to be able to periodically distribute new personal keys via broadcast, so that, after a user is added to the group, no further individual TCP sessions are necessary (provided that the user does not miss more than δ consecutive update messages). This type of technique might also reduce the number of sessions for which a user must be actively revoked, since, if the user possesses personal keys for fewer sessions at any given time, he will not need to be revoked for as many sessions.

8. ACKNOWLEDGEMENTS

The authors would like to thank Russell Impagliazzo for suggestions which improved the definition, and the anonymous reviewers for helpful comments.

9. REFERENCES

[1] S. Berkovits. How to broadcast a secret. In *Advances in Cryptology - Eurocrypt '91, LNCS 547*, pages 536–541, 1991.

[2] C. Blundo, L. F. Mattos, and D. Stinson. Trade-offs between communication and storage in unconditionally secure schemes for broadcast encryption and interactive key distribution. In *Advances in Cryptology - CRYPTO '96, LNCS 1109*, pages 387–400, 1996.

[3] R. Canetti, T. Malkin, and K. Nissim. Efficient communication-storage tradeoffs for multicast encryption. In *Advances in Cryptology - Eurocrypt '99, LNCS 1592*, pages 459–474, 1999.

[4] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., 1991.

[5] C. Dwork, J. Lotspiech, and M. Naor. Digital signets: self-enforcing protection of digital information. In *28th ACM Symposium on Theory of Computing*.

[6] A. Fiat and M. Naor. Broadcast encryption. In *Advances in Cryptology - CRYPTO '93, LNCS 773*, pages 480–491, 1993.

[7] M. Just, E. Kranakis, D. Krizanc, and P. van Oorschot. On key distribution via true broadcasting. In *ACM Conference on Computer and Communications Security*, pages 81–88, 1994.

[8] H. Kurnio, R. Safavi-Naini, W. Susilo, and H. Wang. Key management for secure multicast with dynamic controllers. In *Fifth Australasian Conference on Information Security and Privacy*, pages 178–190, 2000.

[9] D. Liu and P. Ning. Efficient and self-healing key distribution with revocation for tactical wireless networks. In *ACM Conference on Computer and Communications Security*, 2003.

[10] D. McGrew and A. Sherman. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering*, 29(5):444–458, May 2003.

[11] R. Molva and A. Pannetrat. Scalable multicast security with dynamic recipient groups. *ACM Transactions on Information and System Security*, 3(3):136–160, August 2000.

[12] M. Naor and B. Pinkas. Efficient trace and revoke schemes. In *Financial Cryptography, LNCS 1962*, pages 1–20, 2001.

[13] A. Perrig, D. Song, and J. D. Tygar. Elk, a new protocol for efficient large-group key distribution. In *IEEE Symposium on Security and Privacy*, pages 247–262, 2001.

[14] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[15] 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*, pages 241–257, 2002.

[16] D. Wallner, E. Harder, and E. Agee. Key management for multicast: Issues and architectures. In *Internet Draft (work in progress)*, 1998.

[17] C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. In *SIGCOMM '98*, pages 68–79, 1998.

[18] C. Wong and S. Lam. Keystone: A group key management service. In *International Conference on Telecommunications*, 2000.