Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 04-027

New Constructions On Broadcast Encryption and Key
Pre-Distribution Schemes

Scott C.-h. Huang and Ding-zhu Du

June 24, 2004

New Constructions On Broadcast Encryption and Key Pre-Distribution Schemes

Scott C.-H. Huang and Ding-Zhu Du

Abstract—This paper presents various new techniques on secure group communication schemes. We present a new broadcast encryption scheme Arachne, being particularly efficient in multiple revocation, and a node-based key pre-distribution scheme, remedying the key overlapping problem of pool-based schemes. Starting with a detailed analysis on broadcast encryption and group key distribution schemes, we discuss the influence of *join* as well as the feasibility of including it in broadcast encryption schemes by means of performing full updating or overprovisioning.

Index Terms—broadcast encryption, key pre-distribution, security

I. INTRODUCTION

Secure group communication, one of the major problems in network security, is gaining popularity in the Internet community over the last decade. Due to various emerging applications on the insecure Internet, the need of confidentiality, authenticity, as well as other security features was increased more than ever before. In all of these requirements, a fundamental problem: “How to distribute the keys” needs to be dealt with the most, since every type of security-related functionality is involved in cryptographic keys and how to distribute them over the insecure Internet is quite non-trivial.

Traditionally, there are several fundamentally different approaches to deal with key management. **Key pre-distribution** schemes refer to methods whereby a *trusted authority* (TA) distributes secret information to users such that only the specified privileged users can get the group key (by performing computation or decryption). **Contributory group key agreement** schemes require that each member should contribute an equal share of the group key. Multi-party Diffie-Hellman Key Exchange is of this type. **Centralized group key distribution** schemes use a *trusted third party* (TTP) to do all the bookkeeping job. Within this category, we can further divide them into *flat* schemes, in which one group key is maintained, and *hierarchical* schemes, in which some *group security agents* (GSAs) re-lay the communication. In contrast to centralized schemes, **Decentralized group key distribution** schemes also exist, in which the group center’s job is shared by multiple TTPs and the single point of failure problem is somewhat mitigated.

In this paper we focus on two types of secure group communication schemes: *broadcast encryption* and *Key Pre-distribution*. They play a significant role in various applications such as Pay-per-view TV, Video-conferencing, Wireless

and Sensor Networks. We will start with one of the most efficient broadcast encryption schemes SD and our latest scheme Arachne, and then we will carefully investigate the nuance of *join* in both schemes as well as other secure group communication schemes later.

II. BROADCAST ENCRYPTION AND GROUP KEY DISTRIBUTION

A. Overview

Broadcast encryption and group key distribution schemes play a central role in Secure Group Communication protocols. Without facilitating the *join* operation, broadcast encryption schemes find themselves less restricted to *key updating*, thus achieving better performance in doing *multiple revocation*. Group key distribution schemes, on the other hand, provide both join and revocation, thus guaranteeing both *forward* and *backward secrecy*.

a) *Broadcast Encryption Schemes*: The term *broadcast encryption* schemes, first coined by Fiat and Naor [3] in 1993, can be actually viewed as *revocation schemes*. In such schemes, a *trusted authority* (sometimes called the *Group Controller* GC) is assumed to do all the bookkeeping job for a pre-defined set of users. In each session, GC will select a *session key* (sometimes called the *group key*), distribute it to a subset of users called the *privileged users*, and then encrypt all communications with it. The distribution of the session key will certainly be secure against eavesdropping. In other words, the broadcast of the session key will be encrypted, and only those privileged users who hold some secret information will be able to decrypt it. Under such setting, only those in the pre-defined set are eligible to be a privileged user. No other users are allowed to join the group later. The set of users thus defines the *prospective* privileged users, which cannot be changed. Such a strong assumption makes it possible to facilitate a rather hard operation, namely the *multiple revocation*, which was previously thought unfeasible in group key distribution schemes.

b) *Group Key Distribution Schemes*: Group key distribution Schemes provide both *join* and *revocation* operations. There is no pre-defined *prospective user set*, and any user can join the group at any time. Such nice feature makes them more flexible and therefore best suit most applications. Like broadcast encryption schemes, group key distribution schemes have a Group Controller and each user carries some secret information. When membership changes, GC will choose a session key, encrypt it using users’ secret information, and then broadcast it. Only those who remain in the group can use

their secret information to decrypt the message and get the key update. However, unlike broadcast encryption schemes, when a user gets revoked, its secret information will not be used again and whoever holds same piece of the information will get updated. Just because of this extra updating, group key distribution schemes can provide join of new members. And just because of this, group key distribution schemes cannot be efficient to revoke multiple members, since the extra updating of multiple users is usually too slow to be practical.

c) *Multiple Revocation*: As mentioned earlier, group key distribution schemes cannot handle multiple revocation efficiently. Broadcast encryption schemes, on the other hand, can better manage it due to no overhead of providing *join*. Unfortunately, they are not that practical since *join* is needed in most real life applications. Multiple Revocation hence became the major obstacle of Secure Group Communication schemes, and so far there is no good way to handle it. *Multiple Revocation*, possibly being a fundamentally difficult problem, may be intrinsically incompatible with *join*, and the goal of this paper, as well as its main motivation, is centered right around this problem.

B. The Subset Difference Scheme

The SD scheme was originally proposed by Naor *et al* [6], aimed primarily at facilitating efficient multiple revocation. In SD scheme, to revoke r users GC only has to broadcast up to $2r - 1$ messages in the worst case, by far being one of the most efficient schemes. We present this important scheme here, and we will continue to use it as a basis throughout this paper.

d) *Subset Description*: We associate a complete binary tree to the users such that each leaf corresponds to a user. For convenience, we use $v_1, v_2, \dots$ to represent the nodes of the complete binary tree. For all v_i, v_j such that v_i is an ancestor of v_j we consider the subset of nodes in the subtree rooted at v_i but *not* in the subtree rooted at v_j . We denote it by $S_{i,j}$. We will then associate each $S_{i,j}$ with a key $L_{i,j}$. We claim that to revoke any subset $\mathcal{R}$ of size r , $\mathcal{R}$ is actually the union of at most $2r - 1$ subsets of the form $S_{i,j}$. As a result, GC only needs to broadcast at most $2r - 1$ messages to revoke $\mathcal{R}$. We now describe the algorithm of finding such a cover.

e) *Finding the Cover*: Let $ST(\mathcal{R})$ be the Steiner Tree induced by $\mathcal{R}$ and the root. We build the subset collection iteratively to maintain a tree T such that any $u \in \mathcal{U} \setminus \mathcal{R}$ below a leaf of T has been covered. Initially $T = ST(\mathcal{R})$, and then we iteratively remove nodes from T until T is just a single node:

- 1) Find two leaves v_i, v_j in T such that the least-common ancestor v of v_i and v_j does not contain any other leaf of T in its subtree. Let v_l, v_k be v 's children such that v_i, v_j are descendants of v_k, v_l respectively. If there is only one leaf left, make $v_i = v_j$ to leaf, v to be the root of T and $v_l = v_k = v$.
- 2) $v_l \neq v_i$, add $S_{l,i}$ to the collection. Similarly, if $v_k \neq v_j$, add $S_{k,j}$ to the collection.
- 3) Remove from T all the descendants of v and make it a leaf.

The performance of SD can be best described by the following theorem:

Theorem 2.1: Using the Subset Difference method, to revoke r users, GC only has to broadcast at most $2r - 1$ messages.

Proof: We prove that, for any $\mathcal{R}$, SD partitions $\mathcal{U} \setminus \mathcal{R}$ into at most $2r - 1$ disjoint subsets of the form $S_{i,j}$. This is true because every iteration increases the number of subsets by at most two (in step 2) and reduces the number of the Steiner leaves by one (in step 3) except the last iteration that may not reduce the number of leaves but add only one subset. ■

f) *Key Assignments*: To get SD scheme working, we need to assign suitable keys to every subset $S_{i,j}$. The naïve way is assigning different keys $k_{i,j}$ for each set $S_{i,j}$, but it turns out that each user needs to store $O(\log n)$ keys and this is simply not practical. To reduce the storage requirement, we make use of *pseudo-random sequence generators*. We consider those who *triple* the input $G : \{0, 1\}^n \rightarrow \{0, 1\}^{3n}$. Let $G_L(S), G_M(S), G_R(S)$ denote the left, middle, and right third of the output respectively. Now, consider the subtree T_i rooted at v_i and suppose T_i is already given a label LABEL_i . We will label all nodes of T_i as follows: Given a parent being labelled S , its two children are labelled $G_L(S)$ and $G_R(S)$ respectively. Using this labelling method, given any node v_i and its label LABEL_i , we can label all nodes in the subtree of v_i . Let $\text{LABEL}_{i,j}$ be the label of v_j derived from LABEL_i in the subtree T_i . For each $S_{i,j}$, we assign the key $L_{i,j} = G_M(\text{LABEL}_{i,j})$, which is the middle n bits of $G(\text{LABEL}_{i,j})$.

Using the above method, we can quickly get what keys are needed to give to a user u . u should know the key $L_{i,j}$ where v_i is an ancestor of u but v_j is *not*. Note that we only need to consider the case where v_j is a child of v_i since all descendants of v_j can be labelled according to $\text{LABEL}_{i,j}$. Therefore, if v_i is of depth d , exactly $\lceil \log n \rceil - d$ keys needed to be stored. As a result,

$$\sum_{d=0}^{\lceil \log n \rceil} (\lceil \log n \rceil - d) + 1 = \frac{1}{2} \log^2 n + \frac{1}{2} \log n + 1$$

keys needed to be stored at each user. 1 additional key is added for the case of no revocation. Let us summarize SD by the following theorem:

Theorem 2.2: (Performance of SD) To revoke r users, the Subset Difference method requires (1) message length of at most $2r - 1$ keys (2) to store $\frac{1}{2} \log^2 n + \frac{1}{2} \log n + 1$ at a user (3) $O(\log n)$ operations plus a decryption operation to get the re-keying message.

III. ARACHNE: A NEW BROADCAST ENCRYPTION SCHEME

We present Arachne, our new broadcast encryption Scheme here. Arachne, a *Randomized Scheme of Monte Carlo*¹ fashion, was designed to support efficient mass revocation operations.

¹The term *Monte Carlo* is originally used in randomized algorithms, as opposed to the *Las Vegas* type. Monte Carlo algorithms will always guarantee a correct solution, while Las Vegas will not. With the abuse of languages, we use it on our randomized scheme as they share similar ideas.

Despite the use of random functions, its basic idea is actually quite easy and straightforward. GC has l independent (auxiliary) key pools, and before the system starts each user draws a key from each key pool. To revoke a set of users, GC simply uses the keys not held by these users and encrypts the session key update with other (auxiliary) keys. With a certain probability, a remaining user will have a key matching what GC has used to encrypt the session key with. In what follows, we use the language of random functions, which can be thought of as a user drawing a key from the key pool.

A. The Arachne Scheme

g) *Notation*: Let A, B be the source and target set of all of our random functions respectively. Let n be the number of nodes in the network. Assume $|A| = n$, $|B| = m$, and $|S| = l$. We construct a family of random functions $\{f_s|f_s : A \rightarrow B, s \in S\}$, where S is some index set. We sometimes denote it by $(f_s)_{s \in S}$ or even (f_s) when the source, target, and index sets are clear from the context. Our scheme works as follows.

h) The Scheme:

- 1) *Initialization*: Let $(f_s)_{s \in S}$ be a random function family from A to B with index set S . GC generates a set of lm auxiliary keys defined as $\mathcal{K} = \{k_{s,b} | s \in S, b \in B\}$. For each node u_i where $1 \leq i \leq n$, GC secretly gives u_i the set of his auxiliary keys $\mathcal{K}(u_i) = \{k_{s,f_s(i)} | s \in S\}$
- 2) *Re-keying*: If a set of nodes $\mathbf{M}$ are compromised by the enemy, GC must do the re-keying. To remove $\mathbf{M}$ from $\mathbf{U}$ where $|\mathbf{M}| = r \leq m$, GC randomly chooses a key $k_{\overline{\mathbf{M}}}$, encrypts it with every key not belonging to $\mathbf{M}$, separately, and then broadcasts these encrypted messages to all nodes. In short, GC broadcasts

$$\{E_k(k_{\overline{\mathbf{M}}}) | k \in \mathcal{K}, k \notin \mathcal{K}(\mathbf{M})\}$$

- 3) *Decryption*: Each node $u_i \in \mathbf{M}$ uses one of the auxiliary keys $k \in \mathcal{K}(u_i)$ to decrypt $E_k(k_{\overline{\mathbf{M}}})$ and obtain $k_{\overline{\mathbf{M}}}$.

The crucial step in our scheme is step 2, since it is not obvious whether we can find such k 's (i.e. $k \in \mathcal{K}, k \notin \mathcal{K}(\mathbf{M})$). The following theorem shows that we can find such k 's with a really high probability even if l is small.

Theorem 3.1: For each user u_i , the probability of successfully re-keying this user is close to $1 - (1 - e^{-\frac{r}{m}})^l$, i.e.

$$P(\mathcal{K}(u_i) \setminus \mathcal{K}(\mathbf{M}) \neq \emptyset) \approx 1 - (1 - e^{-\frac{r}{m}})^l, \forall u_i$$

Proof: Let's fix u_i . u_i can be re-keyed if and only if $\mathcal{K}(u_i) \not\subseteq \mathcal{K}(\mathbf{M})$. Now we fix s and consider

$$P(k_{s,f_s(i)} \notin \{k_{s,f_s(j)} | j \in \mathbf{M}\})$$

, from which we obtain

$$\begin{aligned} & P(k_{s,f_s(i)} \notin \{k_{s,f_s(j)} | j \in \mathbf{M}\}) \\ &= P(k_{s,f_s(i)} \neq k_{s,f_s(j)} \forall j \in \mathbf{M}) \\ &= \left(\frac{m-1}{m}\right)^r = \left[\left(1 - \frac{1}{m}\right)^{-m}\right]^{-\frac{r}{m}} \approx e^{-\frac{r}{m}} \end{aligned}$$

Therefore,

$$P(k_{s,f_s(i)} \in \{k_{s,f_s(j)} | j \in \mathbf{M}\}) \approx 1 - e^{-\frac{r}{m}}$$

Since there are l such s 's, we have

$$P(u_i \text{ cannot be re-keyed}) \approx \left(1 - e^{-\frac{r}{m}}\right)^l$$

Finally we get

$$P(u_i \text{ can be re-keyed}) \approx 1 - \left(1 - e^{-\frac{r}{m}}\right)^l$$

■

Theorem 3.2: The expected number of messages sent by GC is approximately $lm \cdot e^{-\frac{r}{m}}$

Proof: This theorem can be proved directly from the linearity of expected values. Let's first define what the *Expected Number of Messages sent by GC* means. Let Ω be the sample space of the keys held by $\mathbf{M} = \{u_1, \dots, u_r\}$, and for any $\omega \in \Omega$ we define the *characteristic function* $\chi_{i,s}(\omega)$ as follows:

$$\chi_{i,s}(\omega) = \begin{cases} 1, & \text{if } i \notin \omega_s, \text{ where } \omega_s = \{f_s(u) | u \in \mathbf{M}\} \\ 0, & \text{otherwise} \end{cases}$$

Define the *Number of Messages* with respect to $s \in S$ as the random variable $X_s(\omega) = \sum_i \chi_{i,s}(\omega)$ and the *Number of Total Messages* as $X = \sum_{s \in S} X_s$. The *Expected Number of Total Messages* is thus the expected value, EX , of X . By the linearity of expected values, we have

$$\begin{aligned} EX &= E\left(\sum_{s \in S} X_s\right) = \sum_{s \in S} EX_s = \sum_{s \in S} \sum_{\omega \in \Omega} P(\omega) X_s(\omega) \\ &= \sum_{s \in S} \sum_{\omega \in \Omega} P(\omega) \left(\sum_i \chi_{i,s}(\omega)\right) = \sum_{s \in S} \sum_{\omega \in \Omega} \sum_{i=1}^m P(\omega) \chi_{i,s}(\omega) \end{aligned}$$

Since $\chi_{i,s}(\omega)$ depends solely on f_s , a completely random function independent of s , it follows that $\chi_{i,s}(\omega)$ does not depend on s . We henceforth use $\chi_i(\omega)$ to represent $\chi_{i,s}(\omega)$, and the outmost summation can be simplified as follows

$$\begin{aligned} \sum_{s \in S} \sum_{\omega \in \Omega} \sum_{i=1}^m P(\omega) \chi_{i,s}(\omega) &= \sum_{s \in S} \sum_{\omega \in \Omega} \sum_{i=1}^m P(\omega) \chi_i(\omega) \\ &= l \cdot \sum_{\omega \in \Omega} \sum_{i=1}^m P(\omega) \chi_i(\omega) = l \cdot \sum_{i=1}^m \sum_{\omega \in \Omega} P(\omega) \chi_i(\omega) \\ &= l \cdot \sum_{i=1}^m \left[\sum_{\omega, i \in \omega_s} P(\omega) \chi_i(\omega) + \sum_{\omega, i \notin \omega_s} P(\omega) \chi_i(\omega) \right] \\ &= l \cdot \sum_{i=1}^m \left[\sum_{\omega, i \in \omega_s} P(\omega) \cdot 0 + \sum_{\omega, i \notin \omega_s} P(\omega) \cdot 1 \right] \\ &= l \cdot \sum_{i=1}^m \left[0 + \sum_{\omega, i \notin \omega_s} P(\omega) \right] = l \cdot \sum_{i=1}^m \left(\sum_{\omega, i \notin \omega_s} P(\omega) \right) \\ &= l \cdot \sum_{i=1}^m \left(P(i \notin \omega_s) \right) \end{aligned}$$

Here we view " $i \notin \omega_s$ " as an event and use $P(i \notin \omega_s)$ to denote its probability. The event $i \notin \omega_s$ holds if and only if u_i

can be re-keyed by the keys on line s . Therefore, $P(i \notin \omega_s) = 1 - \left(1 - e^{-\frac{r}{m}}\right)^1 = e^{-\frac{r}{m}}$, obtained by substitute 1 for l . Plugging it in, we finally get

$$l \cdot \sum_{i=1}^m \left(P(i \notin \omega) \right) = lm \cdot e^{-\frac{r}{m}}$$

This theorem is thus proved. ■

i) *Storage Requirement*: In Arachne, each user has to save l auxiliary keys, and GC has to save lm keys.

j) *Security Analysis*: We analyze the security of our scheme in the aspects of *Group Key Secrecy*, *Forward Secrecy*, and *Backward Secrecy*:

- *Group Key Secrecy* : In our scheme, GC generates group keys at random and all communications within the group is encrypted by the group key. The adversary cannot use any feasible means to discover or compute the group keys. As a result, the group key secrecy can be guaranteed.
- *Forward Secrecy* : Forward Secrecy can be achieved by revoking the keys belonging to the revoked nodes. Once a node gets revoked, all keys belonging to him will never be used again. In other words he cannot use his information to decrypt future key updates.
- *Backward Secrecy* : Since this is a broadcast encryption scheme, no join event will happen. The backward secrecy is not in the scope of our discussion.

B. Ensuring Perfect Re-Keying

k) *Monte-Carlo-ized Arachne*: The robustness and performance of the Arachne scheme is best described by Theorems 3.1 and 3.2. However, we cannot ensure that the probability of successful re-keying is always 1. In other words, we cannot guarantee that every legal remaining user will get the session key. To remedy this problem, we can let each user share an individual key with GC. Also, just in case that none of the keys can be used to encrypt with, GC can always encrypt the session key with that individual key. Making users sharing individual keys with GC makes our scheme the *Monte Carlo* type, as every legal user is guaranteed to get the session key update and remain in the group.

l) *Improving the Performance*: We can use the same idea to further reduce the expected number of messages. We can start to use individual keys when most users have already been re-keyed. According to Theorem 3.2, the expected number of total messages is $lm \cdot e^{-\frac{r}{m}}$. If, instead of l 'lines', only x lines ($x < l$) are used, then the expected number of messages is $xm \cdot e^{-\frac{r}{m}}$. Furthermore, the expected number of remaining users not yet re-keyed will be

$$\begin{aligned} & \left[1 - P(\text{successful re-keying}) \right] \cdot (\# \text{ of total remaining users}) \\ &= \left[1 - \left(1 - \left(1 - e^{-\frac{r}{m}} \right)^x \right) \right] \cdot (n - r) \end{aligned}$$

Since these users will be re-keyed by using individual keys, the expected number of total messages GC needs to send will be

$$xm \cdot e^{-\frac{r}{m}} + \left(1 - e^{-\frac{r}{m}} \right)^x \cdot (n - r)$$

Now we view the above expression as a function of x and treat r, m, n as constants. Namely,

$$f(x) = xm \cdot e^{-\frac{r}{m}} + \left(1 - e^{-\frac{r}{m}} \right)^x \cdot (n - r) \quad , 0 \leq x \leq l$$

Then we can always differentiate $f(x)$ to find its minimum. On the other hand, the value l can be decided based on the minimum of $f(x)$ within $x \in [0, \infty)$.

IV. FEASIBILITY OF ADDING JOIN

The lack of *join* oftentimes greatly restricts the applicability of broadcast encryption schemes. It would be much better off to add join to them, even at a higher price. We want to discuss about the feasibility of doing that in the two schemes we have presented so far, namely SD and Arachne. The twin sister of SD, namely the *Layered Subset Difference* scheme, is omitted due to its similar behavior to SD in this case.

m) *To Reuse, Or Not To Reuse?*: The fundamental problem about adding join to broadcast encryption schemes is this question. When nodes are revoked, GC never updates anything other than the session key shared by the remaining nodes and the revoked ones in broadcast encryption schemes, as described in § II. If these pieces of secret information are not updated, then we cannot reuse these 'slots' once occupied by the revoked users. By simply keeping listening after leaving the group, the revoked users can decrypt some later session key as soon as the secret information is ever used again. Conversely, if these pieces of secret information are updated, then the broadcast encryption Scheme will be turned into a new group key distribution scheme, in which the backward secrecy will be ensured by updating the secret information. The goal of this section is to compute how much the updating cost is for the SD and Arachne schemes.

A. Adding Join to SD

We consider the case where one user gets revoked.

Theorem 4.1: SD scheme's updating cost for a single user's revocation is $O(n \log n)$, where n is the number of users in the group.

Proof: Suppose the number of users is an exact power of 2. Then we can use a complete binary tree to represent them. If we fix a degree d , then there are exactly 2^d nodes on that level. Let's pick one node j from that level and assume the nodes on the path from the root to j are $\{a_0, a_1, \dots, a_d\}$, in which a_0 is the root and $a_d = j$. We want to find the common information shared by all nodes of the subtree T_j root at j . Note that if i is an ancestor of j , the common information shared by T_i will be shared by T_j too, as T_j is a subtree of T_i . In our discussion, such information is regarded as T_i 's, thus excluded from T_j 's. In other words, we want to consider the common information shared *exclusively* by T_j but not by any other nodes. Let the j 's sibling be k and consider the keys of $L_{a,b}$ shared by all nodes of T_j . According to § II, a, b should satisfy the following condition:

" a is a common ancestor of all nodes of T_j , and b is the sibling of j ."

The common keys shared exclusively by T_j is thus

$$L_{a_0,b}, \quad L_{a_1,b}, \quad L_{a_2,b}, \quad \dots, \quad L_{a_{d-1},b}$$

, in contrast to all keys known by T_j as follows

$$\begin{array}{ccccccc} L_{a_0,b_1}, & L_{a_0,b_2}, & L_{a_0,b_3}, & \dots, & L_{a_0,b_d} \\ & L_{a_1,b_2}, & L_{a_1,b_3}, & \dots, & L_{a_1,b_d} \\ & & L_{a_2,b_3}, & \dots, & L_{a_2,b_d} \\ & & & \ddots & \vdots \\ & & & & L_{a_{d-1},b_d} \end{array}$$

, where $b_1, b_2, \dots, b_{d-1}$ are the siblings of $a_1, a_2, \dots, a_{d-1}$ and $b_d = b$. From the observation above, we see that the number of keys corresponding to T_j is d , the order of j . On level d , there are 2^d such subtrees. Since GC can multicast a message to a subtree T_j by using a special key S_j shared by all nodes of the subtree (this is a special case of S_{ij} when j is null), the total number of messages GC needs send (measured by the encrypted updating message containing one auxiliary key) is

$$\sum_{d=0}^{\log n} d \cdot 2^d = 2n \log n - \frac{n}{2} + 1 = O(n \log n)$$

This theorem is then proved. $\blacksquare$

According to the theorem, the updating cost for single-user revocation turned out to be prohibitively high, let alone multiple-user revocation. It is simply not practical to do updating in large groups. In the case of LSD, by doing similar computation we can derive similar results, and it is also too high to be practical. The updating cost of LSD is described by the following theorem:

Theorem 4.2: LSD scheme's updating cost for a single user's revocation is $O(\sqrt{n} \log n)$, where n is the number of users in the group.

Since the computation is almost the same as the SD case, the proof is omitted. For the sake of completeness, we list the theorem here.

B. Adding Join to Arachne

Theorem 4.3: If r users are revoked, the expected number of messages GC has to send to update the secret information of the remaining users is approximately $nl(1 - e^{-\frac{r}{m}})$.

Proof: Fix a remaining user u_i and an index (a 'line') s . Consider the probability that GC needs to send u_i a replacement key, i.e.

$$\begin{aligned} P(\text{GC needs to re-key } u_i) &= P(\mathcal{K}_s(u_i) \in \mathcal{K}_s(\mathbf{M})) \\ &= 1 - P(\mathcal{K}_s(u_i) \notin \mathcal{K}_s(\mathbf{M})) \\ &= 1 - P(k_{s,f_s(i)} \notin \{k_{s,f_s(j)} | j \in \mathbf{M}\}) \\ &= 1 - P(k_{s,f_s(i)} \neq k_{s,f_s(j)} \quad \forall j \in \mathbf{M}) \\ &= 1 - \left(\frac{m-1}{m}\right)^r = 1 - \left[\left(1 - \frac{1}{m}\right)^{-m}\right]^{-\frac{r}{m}} \\ &\approx 1 - e^{-\frac{r}{m}} \end{aligned}$$

by applying the same techniques as the proof of Theorem 3.2, we can get the expected number of messages GC has to send is approximately $nl(1 - e^{-\frac{r}{m}})$. $\blacksquare$

Theorem 4.3 tells us that the asymptotic updating cost is $O(n)$ in Arachne. Though better than SD, it is still too costly to be practical in large-scale applications. Actually, almost all broadcast encryption schemes have high key-updating cost. The reason is that the key update needs to satisfy the *Key Independence Condition*, otherwise it will suffer from *Perrig's Attack*. The condition is described as follows:

Definition 4.1: (Key Independence Condition) For any user, it is computationally infeasible to derive any key update from others.

The above condition puts a rather strict limitation on GC's key updating methods. Because of this condition, the key updates of the same user cannot be defined as the function of previous keys and the group key. Detailed discussion will be presented later.

n) Perrig's Attack: The attack was first conceptualized by *Adrian Perrig*. Suppose the previous key and group key can be used to derive the new update in a broadcast encryption scheme, then the security will be seriously undermined if the invalid keys are used again. Consider the following scenario in a broadcast encryption scheme:

Let $\mathbf{M}$ be the set of revoked users. If a remaining user u shares a key (i.e. $\mathcal{K}(u) \cap \mathbf{M} \neq \emptyset$), u just locally replaces each $k \in \mathcal{K}(u) \cap \mathbf{M}$ by $h(k, GK)$, where GK is the group key and h is a pre-defined hash function.

The above method is very efficient, as the updating can be done locally by each remaining user. However, it violates the Key Independence Condition, and an attacker can launch Perrig's Attack as follows:

Suppose u, v are two colliding users not having exactly the same keys (i.e. there is at least one key different). u leaves the group first while v remains in the group. v then sends u the new group key GK as it still has membership. u can use GK to update all of his revoked keys by computing $\{h(k, GK) | k \in \mathcal{K}(u)\}$. Since u has at least one key, say k_0 different from all keys in $\mathcal{K}(v)$, u will possess a freshly updated key k'_0 and use it henceforth even after v quits the group.

Perrig's Attack imposed a strong limitation on updating methods in most broadcast encryption schemes. In most cases, the updating cost will be rather costly and a second-best way to provide join, namely *Overprovisioning*, is more applicable in most cases.

C. Overprovisioning

1) In SD: If the number of users is not an exact power of 2, then the binary tree is not full and there will be some free 'slots' there. In this case, to join a new user, all GC has to do is secretly give that user all necessary information corresponding to one free slot. If the number of users is an exact power of 2, then things will be much more complicated. In this case, GC must "make room" for joining users and the tree must be

expanded. When expansion happens, the size of the tree will be doubled. Let r be the new root, then the old root 0 will become a child of r . Similar to the analysis of Theorem 4.1, GC needs to send the new keys $\{L_{r,t} | t \in T_0\}$ to all remaining users. The following theorem tells us how much the cost is:

Theorem 4.4: To expand the tree, GC has to send exactly n messages, where n is the number of existing users.

Proof: Similar to the proof of Theorem 4.1, to any subtree T_x rooted at node x GC has to send a new key $L_{r,sib(x)}$, where $sib(x)$ is the sibling of x . Since there are n nodes in T_0 and each of which corresponds to a key, GC needs to send n keys in total. ■

Theorem 4.4 tells us that, in order to join a new user, GC needs to send $O(n)$ messages if the tree is full. This rather high cost does not come with surprise: when the binary tree is full and a new node needs to be joined, the cost of join is high due to the cost of restructuring the tree. Fortunately, this won't happen too often, and the *amortized* cost is not too high. In real applications, we normally *overprovision* the keys in order to prevent such tree expansion. The cost of overprovisioning is discussed in the following theorem:

Theorem 4.5: If n is the number of current users. To overprovision n more users, each user has to store $\lfloor \log n \rfloor + 1$ more keys.

Proof: To make room for n more users, the tree must be expanded and the depths of all nodes will be increased by 1. Let r be the new root and 0 be the old one. Each node $u \in T_0$ must store the following new keys: $\{L_{r,sib(x)} | x \in path(u, r)\}$, where $path(u, r)$ is the path from u to r . Since $|path(u, r)| = \lfloor \log n \rfloor + 1$, the theorem is proved. ■

2) In *Arachne*: Quite unlike SD, Arachne has no fixed overhead for overprovisioning. Completely devoid of the idea of 'slots', Arachne has much freedom of fine-tuning the performance. Since its performance totally depends on the parameters l, m, n, r , we can set up their values depending on how much trade-off is needed, thus providing tremendous flexibility for practical applications. Generally speaking, the bigger l is, the more 'room' there is for new users joining. l also represents the storage requirement at nodes. If the nodes can store more keys, Arachne will have better performance, therefore capable of accommodating more new users.

V. KEY PRE-DISTRIBUTION SCHEMES

In this section, we turn our interests to Key Pre-Distribution schemes. In such schemes, no re-keying will be taken place. Before the system starts, suitable (auxiliary) keys will be pre-distributed to each user, and after the system gets booted up, each user will make use of her (auxiliary) keys to communicate with other users. Different from broadcast or multicast communications discussed so far, we will mainly focus on *pairwise* communications hereafter.

A. Pool-Based Schemes

Eschenauer and Gligor [2] used the properties of *Random Graphs* to design a key pre-distribution scheme. The Random Graph model they used is $G(n, p)$, which means that the graph has n vertices and the probability that there exists an

edge between any two nodes is uniformly p (p may depend on n). The notation is listed below:

Symbol	Definition
p	the prob. that an edge exists between two nodes
n	the number of total nodes
d	the expected degree of a node
$G(n, p)$	the random graph of n nodes and edge prob. p
P_{conn}	the probability that the graph is connected

According to Erdős and Rényi [9], if

$$p = \frac{\ln n}{n} + \frac{c}{n}$$

for some real constant c , then

$$P_{conn} = \lim_{n \rightarrow \infty} P[G(n, p) \text{ is connected}] = e^{-e^{-c}}$$

Their scheme works as follows:

- Choose c for desired probability of connectivity P_{conn} such that $P_{conn} = e^{-e^{-c}}$
- Compute p by plugging in $p = \frac{\ln n}{n} + \frac{c}{n}$
- Determine the size of *key pool* P , and the size of users' *key ring* k . P, k should satisfy the following condition:

$$p' := 1 - \frac{((P - k)!)^2}{(P - 2k)!P!} \geq p$$

(this can be done by arbitrarily choosing either P or k and using the above relation to compute the minimum value of the other one)

- After setting up the values P, k , define a key pool of size P and let each user randomly select k keys from the key pool.

The core of the above scheme is to choose P, k so that the probability of sharing (at least) a key between any two nodes is larger than (or equal to) the *threshold value* p , computed by using the random graph theory, thus ensuring that the probability of connectivity is at least as good as our desired value P_{conn} .

Chan *et al* [1] used the same techniques and further generalized Eschenauer-Gligor's scheme. Instead of considering the probability of sharing (at least) a key between two users, they considered the probability of *sharing at least q keys*. Now, P, k are chosen so that the probability of sharing at least q keys between any two users is larger than or equal to the threshold value. This extra overhead makes the scheme more robust against node compromising. Overall, both schemes are good in Sensor Networks, as the storage requirement is low and there is no computation overhead on nodes' side. After pre-distributing the keys to sensor nodes, the network can be maintained by the sensors themselves. The beauty of Random Graph theory makes the scheme easy to implement and quite suitable for sensor networks.

B. A New Key Pre-Distribution Scheme

The main drawback of the previous two schemes is that too many users will share the same key. Given n, P, k , then

probability that a particular key is selected by a particular user is

$$\frac{\binom{P-1}{k-1}}{\binom{P}{k}} = \frac{\left(\frac{(P-1)!}{(k-1)!(P-k)!}\right)}{\left(\frac{P!}{k!(P-k)!}\right)} = \frac{k}{P}$$

Since there are n users in total, the expected number of users sharing this particular key is equal to

$$\frac{k}{P} \cdot n = \frac{kn}{P}$$

The above expression tells us that, on average, $\frac{kn}{P}$ users share a common key, implying that there will be numerous $(\frac{kn}{P})$ -cliques in the graph!

a) *A Node-Based Scheme*: To remedy this problem, we want to design a new scheme ensuring that different keys are used on different links. To implement this, we dropped the idea of key pool and made each node randomly select other nodes to establish links. The scheme works as follows:

- Choose c for desired probability of connectivity P_{conn} such that $P_{conn} = e^{-e^{-c}}$
- Compute threshold p by plugging in $p = \frac{\ln n}{n} + \frac{c}{n}$
- Compute the *desired node degree* $d := p \cdot (n-1)$
- Before the system starts, each node u randomly selects $t = \lceil \frac{d}{2} \rceil$ different nodes $\{v_1, v_2, \dots, v_t\}$ out of the other $n-1$ nodes. GC randomly generates t keys $\{k_1, k_2, \dots, k_t\}$ and then pre-distributes k_i to both u and v_i , for $1 \leq i \leq t$.

b) *Analysis*: Node-Based schemes provides more secrecy between pairs of nodes. No two links being established using the same keys, the security between each pair of nodes has been enhanced. Also, node-based schemes are more robust against node capture, as each captured node only affects those who share a link with it. In pool-based schemes, one node being captured may cause a link physically far away from it to be invalid. As for the connectivity, since both pool-based and node-based schemes use the same random graph model $G(n, p)$, their performance is exactly the same. As a whole, node-based schemes are more secure in pairwise communications and more robust against node capture than pool-based schemes, while maintaining the same connectivity and performance.

VI. RELATED WORK

Fiat and Naor [3], creators of *broadcast encryption* schemes, introduced this idea to handle dynamic membership changes in secure communication groups. Naor *et al* [6] later designed an efficient *stateless* broadcast encryption scheme SD, by far the most efficient broadcast encryption schemes in terms of GC's re-keying message size. SD's unsatisfactory users' storage requirement $O(\log^2 n)$ was reduced by Halevy *et al* [4], making the scheme more practical to large-scale networks. Halevy *et al* also proposed a generalized scheme in [4], allowing more trade-offs. Aside from them, *Perfect Hash Families* were also used to construct different broadcast encryption schemes such as the works of Safavi-Naini *et al* [7] and Fiat-Naor [3].

Group key distribution protocols are mainly built on top of *logical trees*. This idea was first proposed by Wong *et al* [12] and lots of other protocols followed up later, such as [11], [10], [8]. Although asymptotically they all have the same message overhead $O(\log n)$, McGrew and Sherman's scheme [8], with the smallest constant factor, outperforms others. As a whole, all of these schemes achieve good performance when membership change is relatively not sizable.

The concept of probabilistic key pre-distribution was proposed by Eschenauer and Gligor in their pioneering work [2]. Chan *et al* [1] generalized this idea by considering q -compositeness, increasing the security and robustness against node loss. Liu and Ning [5], extending the ideas of [2], [1], later designed a *polynomial-based framework* by using *random subsets* and *grids*. Overall, all key pre-distribution schemes are mainly designed for pairwise communications in sensor networks. They primarily focus on problems caused by the limitation of sensor network such as low computation power, low storage, and unreliable links.

VII. CONCLUSION

We have done a detailed comparison of broadcast encryption and group key distribution schemes, particularly about the join operation. We then have presented our new broadcast encryption scheme Arachne, using the idea of randomized algorithms. Our main contribution is using the nature and beauty of randomized algorithms to make multiple revocation less expensive. We have also discussed the feasibility of adding join to broadcast encryption schemes by doing full updating or overprovisioning. Finally, we have presented our node-based key pre-distribution scheme which solves the problem of key overlapping. Overall, throughout this paper the techniques of randomized algorithms and probabilistic analysis are used extensively, which turned out to be extremely fruitful in modelling numerous problems of this kind.

REFERENCES

- [1] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, May 2003.
- [2] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *9th ACM Conference on Computer and Communication Security*, pages 41–47, Nov. 2002.
- [3] A. Fiat and M. Naor. Broadcast encryption. In *Advances in Cryptology - CRYPTO'93*, 1993.
- [4] D. Halevy and A. Shamir. The LSD broadcast encryption scheme. In *Advances in Cryptology CRYPTO'02*, pages 47–60, 2002.
- [5] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *ACM CCS'03*, 2003.
- [6] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology- CRYPTO'01*, pages 41–62, 2001.
- [7] R. Safavi-Naini and H. Wang. New constructions for multicast re-keying schemes using perfect hash families. New York, NY, USA, 2000. ACM Press.
- [8] A. T. Sherman and D. A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering*, 29(05):444–458, 2003.
- [9] J. H. Spencer. *The Strange Logic of Random Graphs*, volume 22 of *Algorithms and Combinatorics*. Springer-Verlag, 2000.
- [10] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The VersaKey framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications (Special Issue on Middleware)*, 17(9):1614–1631, Aug. 1999.

- [11] D. M. Wallner, E. J. Harder, and R. C. Agee. Key management for multicast: Issues and architectures, 1999.
- [12] C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, Feb. 2000.