KleeQ: Asynchronous Key Management for Dynamic Ad-Hoc Networks

Joel Reardon [1] Alan Kligman [2] Brian Agala [1] Ian Goldberg [1]

[1] David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1
{jreardon,bagala,iang}@cs.uwaterloo.ca
[2] Maplesoft, 615 Kumpf Drive, Waterloo, ON, Canada N2V 1K8
alankligman@alumni.uwaterloo.ca

Abstract

As wireless technology has become ubiquitous, ad-hoc networks have come into wide use. This paper presents a system called KleeQ, which provides secure group communication to users of ad-hoc networks with limited connectivity, such as mobile users communicating over Bluetooth. We provide forward secrecy for this network by asynchronously rotating keys. The need for synchronization or a trusted server is avoided by using a novel method of patching and sealing message blocks; this ensures that the loosely connected group members are able to communicate securely and reliably, even as keys frequently change. The ability to use compromised keys for eavesdropping is limited by negotiating fresh secrets without interfering with the connectivity. KleeQ allows changes in group membership without revealing old messages to new members and vice versa, and automatically forms subgroups when users have lengthy absences. The combination of these features makes KleeQ an ideal system for secure group communication in low-connectivity ad-hoc environments.

1 Introduction

Ad-hoc communication has become increasingly prevalent, particularly with the rise in popularity of wireless networks. These networks present an urgent need for a digital security infrastructure to protect communications sent over insecure channels. Providing useful security to a dynamic group of communicating individuals is of particular interest, because of the challenges associated with group membership and key distribution. A number of approaches to ad-hoc group communications provide encryption and authentication with a focus on key management for rapidly changing groups [3, 11, 13, 14].

Many of these approaches share a common problem: they rely on a trusted server that distributes group keys over existing transport keys. This poses two security risks. First, a trusted server becomes a powerful target for an adversary and a compromise of the key distributor could be disastrous for its users. Second, if one of the transport keys is compromised at a later date, all of the group keys transported with that key are revealed. This can result in a loss of privacy for past communications, illustrating that these systems lack

forward secrecy. Some of these approaches offer forward secrecy in a weaker sense—adding someone to a group does not permit her to read past messages. We define forward secrecy more strongly—past messages should not be readable even by external adversaries that compromise keys in the future. Specifically, after some amount of time any given plaintext message should become irrecoverable to an adversary, even if that adversary compromises the secrets of all participants and also has a copy of all past network transmissions.

We are interested in a network environment with *low connectivity*: nodes do not have long-lived network connections to other nodes and a node may be able to communicate only occasionally with another node in the network. This environment models mobile users communicating opportunistically with a short-range network such as Bluetooth.

The goal of this work is to provide a group communication strategy that removes the need for a trusted server and affords forward secrecy to its users in such an ad-hoc environment, where messages cannot be immediately delivered to the entire group. Additionally, we strive to prevent indefinite eavesdropping even if all the currently used keys were compromised. We have implemented a system called *KleeQ* to achieve this goal. KleeQ is designed for collaboration among a *clique*, which is a set of well-trusted parties where no two clique members are strangers. They share data by updating their peers' view of the total conversation through a procedure called *patching*.

Forward secrecy is obtained by regularly changing a clique secret based on old secrets and previous messages, while taking care to ensure that all clique members—even ones that are available only rarely—can continue to communicate. Key rotation is handled independently for all clique members, so no key information is sent over the network. Old keys, secrets and messages are deleted to limit the impact of a compromised or stolen device. We assume trustworthiness within the clique, and thus do not protect against clandestine attacks that compromise clique members. To prevent indefinite eavesdropping we attach key negotiation parameters to messages sent so that the secret can be renewed independently of any existing keys.

The remainder of this paper is organized as follows. Section 2 discusses related work in group communication. Section 3 discusses the formation of cliques, including secret negotiation. Section 4 presents the method used by KleeQ to send messages between the members. Section 5 presents message blocks, explaining how they are found, verified and removed. Section 6 discusses the security aspects of KleeQ, describing how the secrets of a clique are negotiated, rotated, and managed. Section 7 presents organic subcliques as a method of adapting to settings of limited connectivity or clique members undergoing extended absences, and section 8 concludes.

2 Related Work

Our implementation differs from many existing group communications strategies because of the use of forward secrecy, the lack of a central trusted key distribution server, and the lack of implied full connectivity. Forward secrecy is an important security feature to ensure that a compromised key does not reveal all messages historically sent. The lack of a key server is important for security because a trusted server becomes a powerful target for attackers and places limitations on an ad-hoc network.

Research into key management for groups with dynamic membership have yielded efficient solutions that lack forward secrecy. Wong et al. [14] describe an implemented system that manages group communication by having a trusted server distribute keys, with a focus on scalability for large groups while minimizing encryptions and communications load of the server. When new users wish to join or leave the group, re-keying occurs to prevent the release of old messages; however, each user maintains a unique key with the server that is used to individually encrypt re-keying information. As such, the compromise of this key results in the compromise of transmitted group keys and the loss of forward secrecy.

Similarly, Tseng [13] presents a key tree structure to simplify group inclusion and exclusion. They assign a group member to be the Group Controller, and focus on reducing the storage demand required of the Group Controller. Key rotation occurs only when group membership changes, and re-keying information is sent using existing keys.

To preserve forward secrecy in the two systems above, a key negotiation protocol can be executed to generate ephemeral keys used to transport frequent re-keying information. However the necessity of a central key server or a continually online group controller is a burden we wish to remove with KleeQ.

Kronos [11] is a re-keying approach for secure multicast communications. It uses a domain key distributor and a collection of area key distributors to refresh keys after a fixed time frame. Similar to KleeQ, it uses private state information for the computation of new keys without transmission. However the area key distributors send key information to the group members over existing keys which inhibits forward secrecy.

Huang and Medhi [4] present a keying system for deriving conference keys for subsets of a large group of users based on predistributed key information. Once the information is sent, there is no longer a need for a central server since a group of participants can independently compute a key for communication. Symmetric session keys are generated by one group member and sent to the others using the group key, and as such their scheme has a loss of forward secrecy if the group key is compromised. Group keys can only be changed by having the server distribute new key information.

Mayer and Yung [7] present a series of *expansions* to convert two-party cryptographic protocols into group protocols. They provide an expansion for forward secrecy through the execution of key establishment protocols with a group leader at the beginning of each short lived session. This relies on full connectivity of the group and the discretization of conversation into short-lived sessions with overhead, neither of which are required by KleeQ.

Di Pietro et al. [3] present a system for group communication in a wireless setting relying on a trusted server. It considers dynamic group membership and defines forward and backward secrecy as the inability for excluded members to read future messages and for included members to read previous messages, respectively. Their scheme of key distribution uses a server that distributes information used in conjunction with locally available secret knowledge to compute new keys. This provides forward secrecy against an adversary who compromises the locally available information, but relies on a trusted server and permits indefinite eavesdropping after a compromise.

Borisov et al. [2] have developed a software plug-in to provide forward secrecy and limit indefinite eavesdropping of private instant messenger conversations. It adds key negotiation parameters to each message that is sent, and uses this information to continually derive new keys that are used for as few messages as possible. This approach works well for interactive two-party conversations but its applicability to KleeQ is

limited because of the ad-hoc nature of the group.

3 Clique Formation

A clique is a group of users who communicate in a broadcast manner: all messages addressed to the clique are eventually delivered to all of its members. The *size* of a clique is the number of members in it. Each clique has a *secret* which is updated from time to time, and a clique can be uniquely identified by its name and its secret. Cliques are formed through an interactive process involving key negotiation. We assume that all KleeQ users maintain a public signature key (for example, a DSA key) which is used to authenticate their negotiation parameters, and all KleeQ members who wish to communicate have authentic copies of all relevant public keys. Since clique formation will occur in ad-hoc environments, we allow users to be added at any time. The clique can be constructed gradually until all of the desired members have been added.

One member takes the lead in forming a clique by creating a clique for herself. This is done trivially by selecting a clique name and a random secret. Members are added iteratively, and an authenticated Diffie-Hellman key negotiation [5] takes place at each iteration to ensure that no secrets are ever transmitted over the network. In Diffie-Hellman, both parties in the key negotiation use a secret exponent. A member of the existing clique uses the current clique secret s and sends her invitee g^s. The invitee selects a random secret s' and sends $g^{s'}$ to the inviter. The inviter then sends a message to the existing clique with the name of the new member and his key negotiation parameter $g^{s'}$. When the other members receive the message that a user has been added, they can each compute $g^{s \cdot s'}$ and use the value as the secret for a new clique with the new member. Alternatively, if any of them does not wish to add this new member, he can continue to use the old secret s, which remains a secret from the new member.

Two cliques can merge by having each clique send the exponentiation of g to their current secret and use the negotiated key as their new secret. In fact, additions of a single member can be viewed as merges with one of the cliques having size one.

Removing a member is more cumbersome. While everyone in a clique is fully trusted, it may be necessary to remove a member, particularly if they have been compromised. Naturally, a subset of KleeQ users can opt to form a new clique as above and disregard their previous clique. They may also perform a vote to remove a member from inside the clique. One clique member writes a message containing a motion to remove a member, along with the beginning of a multi-party key negotiation [12]. Each other member can vote in favour of the motion and contribute to the key negotiation, or vote against the motion and avoid key negotiation. After all members vote, the subset of voters who contributed to key negotiation can compute and begin using a new secret based solely on their contributions, and the remaining members will be unable to determine their secret.

4 Conversations

KleeQ manages the messages received by the members of a clique. Communication is achieved through the process of *patching*, where two members of a clique mutually provide missing messages. Each clique

member maintains a *text* for the clique, which is the set of messages he has received. He also keeps track of a *version number* for the clique. This version number is a tuple $v = (v_1, v_2, \ldots, v_n)$ where n is the size of the clique, and v_i is the number of messages in his text authored by the i^{th} clique member.

Algorithm 1 is the patching algorithm used whenever two members of a clique, Alice and Bob, communicate. By exchanging version numbers, each of Alice and Bob can provide any messages the other is missing. Next, we prove the correctness of this algorithm.

4.1 Correctness of the Patching Algorithm

This section provides a proof of the correctness of the patching algorithm; that is, that after running the algorithm, Alice and Bob will have the same text for the clique, which will be the union of their previous texts. This section makes use of a partial ordering of messages, denoted $<$, where $m_1 < m_2$ iff m_1 and m_2 were written by the same author and m_1 was written before m_2.

We can extend this partial ordering to a total ordering $<_L$ in the following way: each clique member maintains a logical *Lamport timestamp* [6], initially 0. When two users patch each other they each set their Lamport time to one plus the maximum of their current Lamport times. When each message is written, it is given a timestamp equal to the author's current Lamport time and then that current time is increased by one. We note that Lamport times are always nonnegative integers. Then if message m' was written by clique member U and message m was written by clique member V, we say $m' <_L m$ iff the timestamp on m' is less than that on m, or they are equal, and $U < V$ in some canonical ordering of the clique members, such as an alphabetical ordering of their names.

We say the *total Lamport time* of a clique is the sum of the current Lamport times of each of its members. We note that creating messages and patching other clique members always strictly increases the clique's total Lamport time.

We first show that no text ever contains gaps in messages by a single author.

Lemma. *Whenever a clique member's text contains a message m, it also contains all messages m' such that $m' < m$.*

Proof. When the clique is created, all clique members' texts are empty, and so the statement is vacuously true when the total Lamport time is 0.

Suppose the statement is false when the total Lamport time is t, and let this be the earliest time it is false; that is, the statement is true for all times less than t. (Since Lamport times are nonnegative integers, we can always do this.)

At total Lamport time t, let V be a member of some clique, and suppose m is a message in V's text, m' is a message not in V's text, but $m' < m$. There are two ways for m to have been added to V's text: either V wrote the message m, or else V received m during a patching operation with some other clique member W. Note that W may or may not be the author of m.

But if m' is not in V's text, then V is not the author of m', so by the definition of $<$, V is also not the author

Algorithm 1 The Patching Algorithm

Alice

1. Alice sends her version number $v^A = (v_1^A, \ldots, v_n^A)$ to Bob

Bob

1. Bob computes the difference between his version number and Alice's:

$$\begin{aligned} v^\Delta &= v^B - v^A \\ (v_1^\Delta, \ldots, v_n^\Delta) &= (v_1^B - v_1^A, \ldots, v_n^B - v_n^A) \end{aligned}$$

2. $\mathcal{R} \leftarrow \emptyset$ (The return message)

3. **foreach** i from 1 to n:

   if $v_i^\Delta > 0$: (Alice is missing messages from author i)

   Add the v_i^Δ most recent messages in Bob's text authored by the i^{th} clique member to $\mathcal{R}$

4. Bob sends his version number and $\mathcal{R}$ to Alice

Alice

1. Alice adds messages from Bob's $\mathcal{R}$ to her text.

2. As above, Alice computes the difference between her (new) version number and Bob's version number: $v^\Delta = v^A - v^B$.

3. As above, Alice computes the set of Bob's missing messages $\mathcal{R}$ and sends it to Bob.

Bob

1. Bob adds the messages Alice has sent to his text.

of m. Therefore, V received m from W in a patching operation, and m and m' were both written by some author i. Before that patching operation, the total Lamport time was less than t, so since m was in W's text before the patch, so were all messages written before m by author i. Therefore, before the patch, we must have had $v_i^V < v_i^W$, V's text contained the first v_i^V messages written by author i, and W's text contained the first v_i^W messages written by author i. Then during the patch, $\mathcal{R}$ will contain exactly those messages from author i which W had but V did not. Therefore either m' was already in V's text before the patch, or it was in $\mathcal{R}$, and was added to V's text during the patch. Either case contradicts m' not being in V's text after the patch, and the proof is complete. □

Theorem 1 (The Patching Theorem)**.** *Let $v^A = (v_1^A, \ldots, v_n^A)$ and $v^B = (v_1^B, \ldots, v_n^B)$ be Alice and Bob's version numbers before executing the patching algorithm. After executing the algorithm, they will each have version number $v^{AB} = (v_1^{AB}, \ldots, v_n^{AB}) = (\max(v_1^A, v_1^B), \ldots, \max(v_n^A, v_n^B))$, and each of their texts will consist of the first v_i^{AB} messages written by each member i.*

Proof. As a consequence of the Lemma, we know that whenever a clique member has version number $v = (v_1, \ldots, v_n)$, his text contains exactly the first v_i messages written by author i, for each i. Therefore, all we need to do is compute Alice and Bob's version numbers after the patch, and show that they each equal v^{AB}.

But this is immediate: for each author i, if $v_i^B > v_i^A$ then Bob will send Alice exactly $v_i^B - v_i^A$ messages from author i, so she will end up with v_i^B such messages, and similarly if $v_i^B < v_i^A$. □

5 Blocks and Verification

Despite having a mathematical proof of the patching algorithm's correctness, in practice we still wish to verify that the messages were transmitted correctly; system errors may occur, network errors are inevitable, and an adversary might attempt to insert fake messages. We wish to verify that the messages received by one clique member are the same as those being received by the others. This is complicated, however, by the ad-hoc nature of the group: messages are transmitted between peers only when they happen to communicate. Recalling the total ordering of messages $<_L$ from section 4, we can arrange all messages into a linear sequence, but it may be the case that newer messages (ordered by Lamport time) may arrive before some older messages. Note, however, that not only does $<_L$ respect causality (if message r is a response to message m, then necessarily $m <_L r$), but it can never be the case that a clique member receives a response to a message before receiving the original message.

To verify that a set of messages is correct, KleeQ divides the sequence of received messages into consecutive *blocks*, and *seals* a block when it is certain that no message is missing within the block. It does so in a deterministic way that guarantees that all clique members will find the same messages when each calculates his or her own sealed blocks. The most recently received messages, which are not part of any sealed block, are called the *tail*. The *sealable set* is a prefix of messages in the tail such that no message is missing from the sealable set. Blocks are found and sealed (and thus removed from the tail) by examining the sealable set.

Since sealed blocks will be found identically for all users, they can be used to verify the equality of the messages they contain. KleeQ users compare cryptographic hashes of the contents of the blocks they seal.

Algorithm 2 The Block Finding Algorithm

Inputs

- Clique $\mathcal{C}$ with a sequence $\mathcal{M}$ of unsealed messages

Output

- The prefix of $\mathcal{M}$ which forms a sealable block, if any

Variables

- t: Total number of uniquely seen authors
- p_i: Computed position of the i^{th} member's last message
- j: Message iterator
- $\mathcal{S}$: The sealable set

Procedure

1. $t \leftarrow 0$
2. $p_i \leftarrow$ **null** $\forall i \in \mathcal{C}$
3. $j \leftarrow$*last-message*$(\mathcal{M})$
4. **while** $j \geq_L$ *first-message*$(\mathcal{M})$:
   - $i \leftarrow Author(j)$
   - **if** $p_i =$ **null**:
     - $p_i \leftarrow j$
     - $t \leftarrow t + 1$
     - **if** $t = |\mathcal{C}|$:
       - $\mathcal{S} \leftarrow (\mathcal{M}_1, \ldots, \mathcal{M}_j)$
       - **break while**
   - $j \leftarrow$ *previous-message*(j)
5. **if** $t \neq |\mathcal{C}|$: **return** $\emptyset$
6. $t \leftarrow 0$
7. $p_i \leftarrow$ **null** $\forall i \in \mathcal{C}$
8. $j \leftarrow$*first-message*$(\mathcal{S})$
9. **while** $j \leq_L$*last-message*$(\mathcal{S})$:
   - $i \leftarrow Author(j)$
   - **if** $p_i =$ **null**:
     - $p_i \leftarrow j$
     - $t \leftarrow t + 1$
     - **if** $t = |\mathcal{C}|$:
       - $end \leftarrow \max_i(p_i)$
       - **return** $(\mathcal{S}_1, \ldots, \mathcal{S}_{end})$
   - **else**: $p_i \leftarrow j$
   - $j \leftarrow$*next-message*(j)
10. **return** $\emptyset$

If the hashes differ between users, then an error has occurred in transmission and the blocks can be compared for resolution. When all users have verified the contents of a block, the block can be deleted since no clique member will need to request a message it contains. In section 6 we will see that blocks also play an important role in key rotation and forward secrecy.

5.1 The Block Finding Algorithm

Algorithm 2 presents the block finding method that is used to seal blocks from the tail. After each patching, clique members repeatedly run this algorithms to find and seal blocks until the algorithm returns the empty block $\emptyset$.

The procedure is broken into two phases. The first phase (steps 1–5) returns the sealable set from the tail and the second phase (steps 6–10) returns the first block in the sealable set. A block is defined as the smallest set of sequential messages, starting from the beginning of the sealable set, such that every clique member has authored at least one message in the block. We choose this definition so that we usually find at most one sealable block after each expansion of the sealable set; sealing multiple blocks with no intervening communication provides no additional benefit for the increased cost.

Figure 1 illustrates this algorithm. In Figure 1(a) we have received 22 messages from our clique. Each message is assigned one of four colours, indicating which of the four clique members authored it. These messages are sorted in the diagram according to increasing Lamport time; that is, by the total ordering $<_L$. In Figure 1(a) we see that we have previously found and sealed one block of messages, and now we run the algorithm to try to find the next block.

We run the first phase of the algorithm to find the sealable set, pictured in Figure 1(b). This is the longest prefix of the tail with the property that we have received a message from every author at or to the right of the rightmost element of the sealable set. This ensures that there are no messages yet to be received that have Lamport times within our sealable set. Note that this is not the case outside our sealable set; we may yet receive messages from the dark grey author with a Lamport time before the rightmost message from the light grey author, for example.

The second phase of the algorithm examines only the sealable set, and looks for the first block: the shortest prefix of the sealable set that contains a message from every author. The result is shown in Figure 1(c). After this returned block is sealed, the system will be in the state pictured in Figure 1(d). Note that no further blocks are available to be found at this point; the block finding algorithm will locate more blocks only after we receive more messages in our next patching.

Correctness of the Block Finding Algorithm

Correctness is shown in two parts: proving that a sealed block has no missing messages, and proving that all clique members will find the same blocks. We assume that there have been no errors in transmission or fraudulent messages.

Lemma. *The sealable set $\mathcal{S}$, if nonempty, contains no missing messages.*

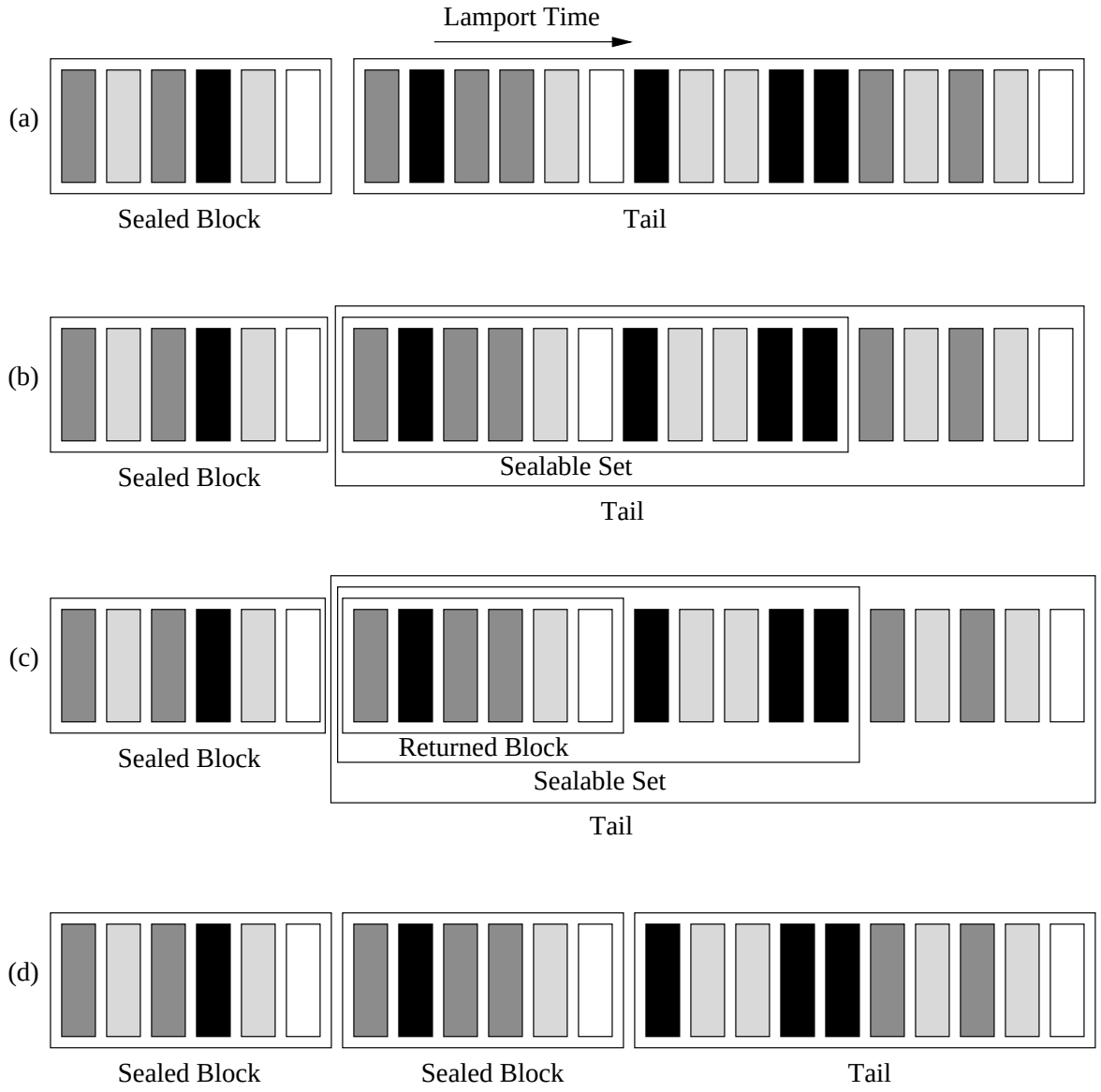

Figure 1: An illustration of the block finding algorithm. Each shaded rectangle represents a message; shades correspond to unique authors. (a) illustrates the input to the algorithm; (b) is the result of the first phase; (c) is the output of the algorithm; (d) is the state of the system after the returned block is sealed.

Proof. After step 5, $\mathcal{S}$ will be the longest prefix of $\mathcal{M}$ with the property that for every message s in $\mathcal{S}$, and for all authors i, there is a message $m_i \geq_L s$ in $\mathcal{M}$ written by author i.

Suppose user W's sealable set $\mathcal{S}$ contains a missing message m by an author V; that is, V wrote a message m with $m <_L \mathcal{S}_{|\mathcal{S}|}$ but W has not yet received m. We know W has received some message $m_V \geq_L \mathcal{S}_{|\mathcal{S}|}$ also written by V. But by the patching theorem, we know that W must also have received all messages written by V before m_V, including m, a contradiction. □

Theorem 2. *Blocks returned by the block finding algorithm contain no missing messages.*

Proof. If the block finding algorithm returns a sealable block, it is a prefix of the sealable set. By the Lemma, this block will contain no missing messages, as required. □

Lemma. *If the second phase of the algorithm finds a block in the sealable set $\mathcal{S}$, then it would find the same block in any sealable set $\mathcal{S}'$ for which $\mathcal{S}$ is a prefix of $\mathcal{S}'$.*

Proof. The second phase of the algorithm begins examining the first message in the sealable set and continues until it finds at least one message from each author. Therefore, if it successfully finds a block in $\mathcal{S}$, it would find that same block in $\mathcal{S}'$, which starts with a copy of $\mathcal{S}$. □

Theorem 3. *If a clique member V has found and sealed a sequence of blocks $\mathcal{B}^V = \mathcal{B}_1, \ldots, \mathcal{B}_\ell$, then each other member W of the clique will have found and sealed a sequence of blocks $\mathcal{B}^W$ where either $\mathcal{B}^V$ is a prefix of $\mathcal{B}^W$, or $\mathcal{B}^W$ is a prefix of $\mathcal{B}^V$.*

Proof. By induction on ℓ, the number of blocks in $\mathcal{B}^V$.

Base case: $\ell = 0$, so $\mathcal{B}^V$ is empty, and is therefore a prefix of $\mathcal{B}^W$.

Inductive case: Assume the statement is true whenever $\mathcal{B}^V$ is of length $\ell - 1$. Suppose $\mathcal{B}^V$ is of length ℓ, and let W be any other member of the clique. Let $\mathcal{B}^V_*$ be the first $\ell - 1$ blocks of $\mathcal{B}^V$. By the inductive hypothesis, we know that if the length of $\mathcal{B}^W$ is at most $\ell - 1$, then it is a prefix of $\mathcal{B}^V_*$, and so also of $\mathcal{B}^V$. Otherwise, the first $\ell - 1$ elements of $\mathcal{B}^W$ are a copy of $\mathcal{B}^V_*$, and all that remains to be shown is that the ℓ^{th} element of $\mathcal{B}^V$ is the same as the ℓ^{th} element of $\mathcal{B}^W$.

After finding and sealing the same $\ell - 1$ blocks, V and W's sealable sets $\mathcal{S}^V$ and $\mathcal{S}^W$ will be nonempty (since they are each able to find and seal another block), and so will each start with the same message. Since sealable sets cannot be missing messages, we must have that one of $\mathcal{S}^V$ and $\mathcal{S}^W$ is a prefix of the other, and so by the Lemma, the block finding algorithm will return the same ℓ^{th} block in either case. □

As a result of this theorem, we see that as long as all clique members continue to both write messages and communicate, they will all seal off the same sequence of blocks of messages. To solve the problem of clique members not writing messages, "dummy" messages are automatically authored for clique members when they patch other members. To handle the case where members do not communicate for a long time, KleeQ uses a mechanism called *organic subcliques*, which we will discuss in section 7.

5.2 Block Verification

The previous section shows that all members of a clique will be able to find the same set of blocks regardless of the sequence of patchings that distributed the messages. This useful property allows members to independently compute hash digests of the resulting blocks to verify their equality.

Each block is given a sequential number equal to the number of blocks sealed before it. The concatenation of the number and the contents of the messages in the block is hashed to produce the fingerprint of the block. Two clique members can verify that these fingerprints are equal to be certain that no errors have occurred during transmission.

If two fingerprints disagree, then perhaps a message has been forged and inserted into the text. Both users would send the entire block to each other's devices. Although not yet implemented, it is expected that the automated verification mechanism would then cease and the users would be prompted that some malicious activity may have occurred. They would be able to inspect the differences between the two blocks for manual resolution.

5.3 Safe Deletion

Safe deletion is the process by which clique members can remove old messages with the assurance that all clique members have been fully patched up to that point. Clique members keep track of block verifications made with other clique members. Each user maintains a list of other clique members, and associates with each member the highest block number that has been verified with that person. When a block has been verified across all members of the clique, then it can be safely deleted. Once a block has been deleted by all clique members it cannot be recovered, which is part of how KleeQ achieves forward secrecy. The other part is key management, which we discuss next.

6 Key Management

It is critical for our application that clique conversations are encrypted and authenticated. We assume that all clique members have authentic copies of each others' public signature keys. Long-lived public encryption keys are not used, because any message encrypted with such a key would lack forward secrecy. As we saw in section 3, clique members use these signature keys during clique formation to execute a multi-party key negotiation protocol that provides each clique member with a common secret. In addition, each clique has a public clique name that is known to all clique members. The secret and the clique name are used to derive an initial key for encryption and message authentication.

6.1 Key Negotiation

As discussed in section 3, key negotiation is used to provide all clique members with an initial secret. To prevent indefinite eavesdropping in the event that the secret is compromised, it is important to renew the

secret regularly.

Multi-party Diffie-Hellman [12] is used to generate new secrets for the clique. In this protocol, each member of the clique receives a set of key negotiation parameters and contributes signed parameters of their own. This key renewal is done in an ad-hoc manner, where clique members attach key negotiation parameters to messages they write. KleeQ will examine the current state of key negotiation based on received messages, and if a clique member has not yet contributed to this round she appends her own contribution. When all members have contributed, the last writes a message containing a set of parameters from which only contributors can compute the final key. When the block containing that message is sealed, KleeQ will notice this message is present and begin using the new secret.

6.2 Message Authentication

Message authentication is needed for all communications that occur between clique members, such as comparing version numbers, sending patches, and verifying old blocks. KleeQ provides group authentication through the use of a *Message Authentication Code (MAC)* [8] based on the current shared key and the message contents themselves. These MACs ensure that the original message has not been modified by an adversary through transmission. We use SHA256-HMAC [1], a MAC based on adding keying to the standard SHA-256 hash function [9].

We should note that the use of a MAC only authenticates a message as coming from *some* clique member, not any member in particular, but this is acceptable in our scenario of trusted clique members, and forgeries will eventually be discovered during the block sealing process in any event. Similarly, it does not force non-repudiation on KleeQ users. If a clique wishes to have non-repudiation for all messages in the conversation then it is trivial to use the existing public keys to sign messages.

6.3 Key Derivation and Rotation

The MAC function is also used to derive an initial key and rotate old keys into fresh ones. When a clique is formed, clique members are provided an identical set of information: a clique name $\mathcal{N}$, and a shared secret s. The initial key k used for communications is derived in the following manner: $k \leftarrow MAC_s(\mathcal{N})$; that is, the MAC for the clique name keyed by the secret.

By rotating keys after sealing each block, KleeQ messages gain forward secrecy. Since blocks are identical for all clique members, new keys can be derived from old keys and the block contents, and can be independently computed by all members. The process for key rotation is as follows:

$$\begin{aligned} s &\leftarrow MAC_k(s) \\ k &\leftarrow MAC_s(block\,contents) \end{aligned}$$

The first line changes the shared secret for each clique member. If the key for a message was compromised, an adversary would still need to know the current secret to generate the key used following key rotation.

The second line rotates the current key by finding the MAC of the current block, keyed by the current secret. When a new secret is computed through negotiation parameters attached to messages, it is used as the new secret instead of computing $MAC_k(s)$.

Old keys should be kept in the system until a user is certain that no other clique member would still be using that key. When patching requests arrive encrypted with an old key, the member with a newer key can update the other user using the old key. Then both members will now be able to seal off the same blocks and can communicate with the new key. When a user is certain all members of a clique have sealed off a block, he must securely remove keys and secrets derived before that block.

6.4 Message Format

In addition to the usual encryption and authentication, we must deal with an addressing issue. It is possible for clique members to get messages destined for other cliques, especially when using a local area broadcast transport protocol such as Bluetooth or Wi-Fi. Moreover, key rotation can result in a variety of relevant encryption keys in a single clique, and the user would need to efficiently know which key to use. It is important for clique members to quickly identify messages that are related to a clique in which they are a member, and also to identify which key to use for message decryption. Finally, it is preferable to encrypt the name of the clique and current key iteration for the privacy of KleeQ users and to hinder targeted eavesdropping.

An address tag, c_{id}, is provided at the beginning of any communication between clique members. It uniquely refers to the clique for which this message is destined, and identifies the key used to encrypt the communication. The tag consists of the MAC for the clique name $\mathcal{N}$ keyed by the key k being used to encrypt remainder of the message: $c_{id} \leftarrow MAC_k(\mathcal{N})$. Thus, the entire communication for a message m is the address, payload and MAC: $MAC_k(\mathcal{N})||E_k(m)||MAC_k(\mathcal{N}||E_k(m))$. To prevent replay attacks, the message contains the names of the sender and the recipient, as well as the sender's current Lamport time.

Each time a clique key is rotated, the clique member computes the address tag c_{id} and adds it to an internal look-up table of keys indexed by the tag. When a message arrives, users can quickly consult the table to determine if they are capable of decrypting the message, and which key to use for decryption. When keys are no longer needed they are removed from the table to prevent an attacker from accessing old keys through a compromised device. Users can maintain a common table of address tags and keys for all cliques in which they are involved.

Keys are added to the table when they are computable by each clique member individually and independently of the other members. This relieves the need for transporting keys over the network or a trusted server to indicate when to begin using new keys. Since the block's contents are used to generate new keys, when two clique members communicate with a new key, they have performed a de facto validation of the block from which the key was derived. Moreover, because of the cumulative derivation of keys, they have additionally validated all earlier blocks. This swift validation allows for old blocks to be quickly removed from memory.

It is possible that an adversary could observe various clique members sending messages with the same tag header. While the header would only be the same for messages delivered within a single block of the text, an adversary could use this information to infer the set of devices that are involved in a clique. To protect

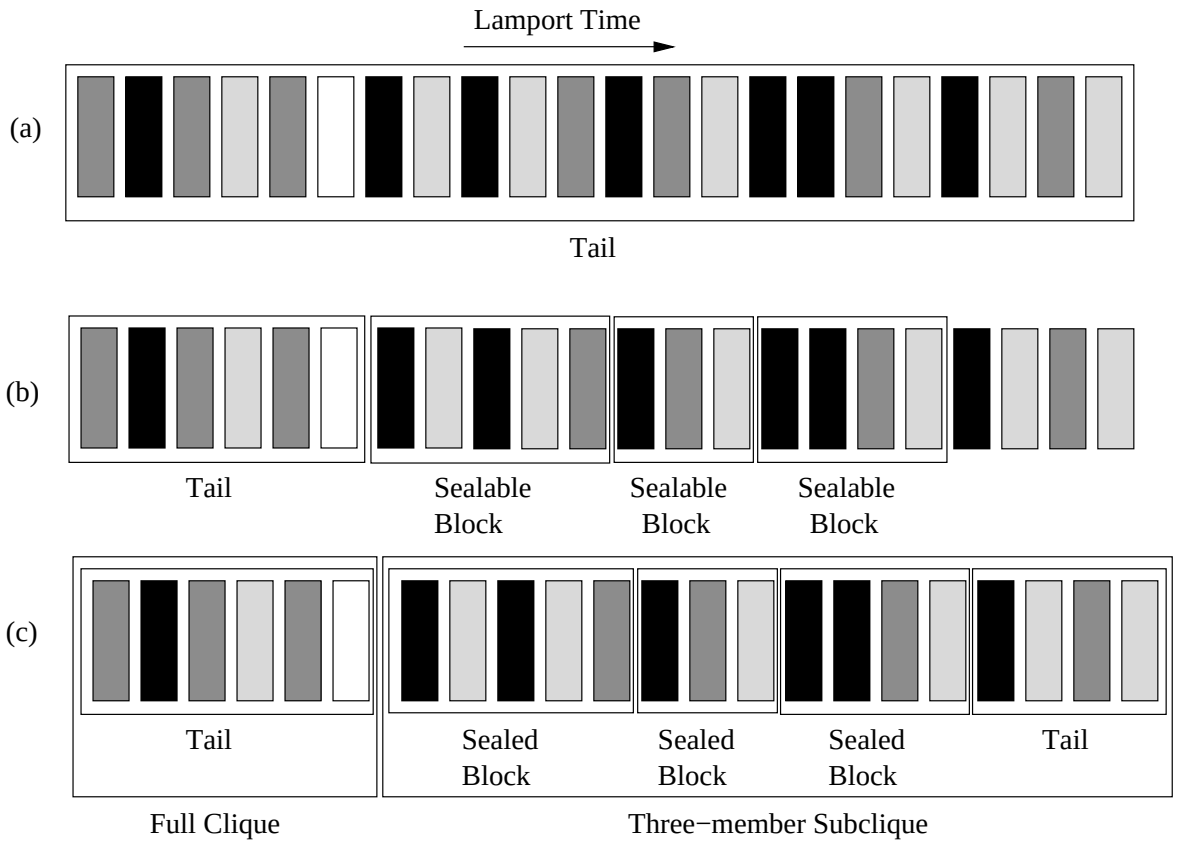

Figure 2: An illustration of the organic subclique algorithm. Each shaded rectangle represents a message; shades correspond to unique authors. (a) illustrates the input to the algorithm; (b) is the discovery of a subclique; (c) is the state of the system after the smaller subclique is used to seal more blocks.

against this, we can create multiple clique *aliases* by appending a small number after the clique name before calculating the MAC to produce different address tags that correspond to the same key. Since the list of tags and keys are managed in a hash table, the only disadvantage is that more space is required to store this extra information.

7 Organic Subcliques

In large cliques with low connectivity it is likely that some subset of members may repeatedly patch each other for a duration without patching other members of the clique, as seen in Figure 2(a). Such a subset of users will wish to seal off blocks so that key rotation can occur without waiting for the missing members. Users may also take a vacation and be unable to participate in patching, while the remaining KleeQ users will want to verify their conversation in the interim.

We consider each clique member as being involved in two cliques: the full clique, and the reduced clique based on current connectivity. A user can construct their reduced clique by determining how many blocks could be sealed if the subclique consisted of only the authors in some subset, as seen in Figure 2(b). If the number of blocks is above some threshold, then a control message can be sent where a clique member indicates he can seal off n blocks with reduced clique $\mathcal{C}_r \subsetneq \mathcal{C}$, and asks if any other clique member in $\mathcal{C}_r$ has sent messages from the last n blocks to clique members outside $\mathcal{C}_r$. Once all members of $\mathcal{C}_r$ vote to seal off the block, then they can remove the block, verify the old block, and perform key rotation within this new group, as seen in Figure 2(c). This entire procedure can be performed automatically with a parameterized threshold of sealable blocks that triggers a voting message.

When clique members outside of $\mathcal{C}_r$ resume patching, the key that was used before the creation of the subclique can resurface. However, the messages assigned to the subclique will not be available to the excluded members of the original clique; key rotation and forward secrecy may well cause subclique members to discard those messages before excluded members return. This gives cliques an organic quality, where subcliques naturally follow the observed connectivity of the clique members.

8 Conclusion

Mobile users with local communication form an ad-hoc network with low connectivity. This paper presents a system called KleeQ, which provides secure group communication to users of such a network, even where the networking environment precludes the use of an always-available trusted server. KleeQ provides forward secrecy by regularly rotating keys, and uses a novel method of patching and sealing message blocks to ensure that the loosely connected group members are kept in sync. KleeQ allows group membership to change without revealing old messages to new members, and vice versa, and can automatically form subgroups when users have not been heard from in some time.

Our prototype implementation of KleeQ is available for download on SourceForge [10].

References

[1] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology—CRYPTO '96, Lecture Notes in Computer Science 1109*, pages 1–15. Springer-Verlag, 1996.

[2] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-Record Communication, or, Why Not To Use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society (WPES 2004)*, pages 77–84. ACM Press, 2004.

[3] Roberto Di Pietro, Luigi V. Mancini, and Sushil Jajodia. Efficient and Secure Keys Management for Wireless Mobile Communications. In *Proceedings of the second ACM international workshop on Principles of mobile computing (POMC '02)*, pages 66–73. ACM Press, 2002.

[4] Dijiang Huang and Deep Medhi. A Key-Chain-Based Keying Scheme For Many-to-Many Secure Group Communication. *ACM Transactions on Information and System Security*, 7(4):523–552, 2004.

[5] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In *Advances in Cryptology—CRYPTO 2003, Lecture Notes in Computer Science 2729*, pages 400–425. Springer-Verlag, 2003.

[6] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[7] Alain Mayer and Moti Yung. Secure Protocol Transformation via "Expansion": From Two-party to Groups. In *Proceedings of the 6th ACM conference on Computer and communications security (CCS '99)*, pages 83–92. ACM Press, 1999.

[8] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*, chapter 9: Hash Functions and Data Integrity. CRC Press, 1996.

[9] National Institute of Standards and Technology. Secure Hash Signature Standard. Federal Information Processing Standards Publication 180-2, 1 August 2002.

[10] Joel Reardon and Alan Kligman. KleeQ project on SourceForge. http://sourceforge.net/projects/kleeq/.

[11] Sanjeev Setia, Samir Koussih, Sushil Jajodia, and Eric Harder. Kronos: A Scalable Group Re-Keying Approach for Secure Multicast. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 215–228, May 2000.

[12] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 31–37, 1996.

[13] Yuh-Min Tseng. A scalable key-management scheme with minimizing key storage for secure group communications. *International Journal of Network Management*, 13(6):419–425, 2003.

[14] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure Group Communication Using Key Graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, February 2000.