

# Chord-based Key Establishment Schemes for Sensor Networks

Fan Zhang, Zhijie Jerry Shi, Bing Wang

Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269

**Abstract**—Because of limited resources at sensor nodes, sensor networks typically adopt symmetric-key algorithms to provide security functions such as protecting communications between nodes. In order to use symmetric-key algorithms, two nodes need to establish a secret session key first. In this paper, we propose a novel chord-based key establishment (CBKE) protocol that allows any pair of nodes in a sensor network to establish a secret session key. CBKE is a generalized deterministic key establishment scheme that provides great flexibility for balancing memory overhead, reliability, and communication cost. We analyze the properties of CBKE and explore the performance trade-offs using simulation.

## 1. Introduction

As sensor networks are adopted in more and more applications, the security of sensor networks becomes increasingly important because sensor nodes are deployed in the field and communicate through wireless channels. Many security goals are achieved through cryptographic algorithms. Major types of cryptographic algorithms include symmetric-key and public-key algorithms. Typically, sensor networks adopt symmetric-key algorithms since public-key algorithms are computationally expensive while sensor nodes have limited computational capability and battery energy. To use symmetric-key algorithms, two sensor nodes need to agree on a secret key first. This process is called *key establishment* (or *key exchange*). While public-key algorithms are used to perform key establishment in systems that have more resources, it is desirable to perform key establishment in sensor networks with symmetric-key algorithms, for the same reasons described earlier.

In this paper, we focus on *pairwise* key establishment that allows any two nodes in the network to agree on a secret key, since pairwise communications are the most common cases in sensor networks [13]. Existing pairwise key establishment schemes can be broadly classified into the following four major categories: *trusted-server-based schemes*, *random key pre-distribution schemes*, *deterministic key pre-distribution schemes*, and *location-based schemes*. Trusted-server-based schemes (e.g., [11])

use a trusted server as an arbiter to establish pairwise keys between two nodes. Random key pre-distribution schemes (e.g., [4], [5]) preload a set of keys chosen from a global key pool into each node. Two neighboring nodes share a common key with a certain probability. Two nodes not pre-sharing a key establish keys by finding a path in the induced key-sharing graph (where two nodes are connected if they share a key). The performance of random key pre-distribution schemes can be improved by using multiple key spaces (e.g., [3], [8]). Deterministic key pre-distribution schemes pre-distribute keys in a deterministic manner. One extreme form of deterministic key pre-distribution is full pairwise key pre-distribution in which every pair of nodes share a unique key and each node has to store many keys in its memory. The other extreme is that all nodes preload a single key. PIKE [1] is between these two extremes. It considers the node ID space as a logical grid. Any two nodes in the same row or column pre-share a unique key; nodes not pre-sharing a key establish keys through intermediate nodes. Location-based schemes (e.g., [2], [9], [14]) take advantage of location information to improve key establishment.

Several other types of key establishment schemes are proposed recently. For example, [10] proposes a self-configuring scheme that adapts to the post-deployment network topology and requires no location information beforehand; [13] proposes a random perturbation-based pairwise key establishment scheme that utilizes multiple perturbed polynomials to establish keys between any pair of nodes directly.

In this paper, we propose a novel chord-based key establishment (CBKE) protocol for pairwise key establishment. CBKE is a deterministic key pre-distribution scheme. It is very flexible and achieves a wide range of performance trade-offs among memory overhead, reliability, and communication cost. As we shall see, existing deterministic key distribution schemes, including full pairwise pre-distribution, single network-wide key scheme, and PIKE, are special cases of our scheme. By choosing parameters properly, CBKE can meet various design goals. For example, in networks that require low memory overhead, the proposed scheme can reduce the number of preloaded keys to  $\log_2 N + 1$ , where  $N$  is the

number of nodes in a network.

The rest of the paper is organized as follows. Section 2 presents the CBKE scheme. We discuss its properties in Section 3 and study performance trade-offs in Section 4. After discussing several improvements in Section 5, we conclude the paper in Section 6.

## 2. Chord-based Key Establishment Scheme

In this section, we first present the basic CBKE scheme and then describe a generalized scheme.

### 2.1. Basic CBKE scheme

Since CBKE employs deterministic key pre-distribution, we start with the key pre-distribution process. We then describe the key establishment process and a mechanism to improve reliability.

**2.1.1. Key pre-distribution in basic CBKE:** Consider a sensor network of  $N$  nodes. Node  $i$  is assigned an ID of  $i$ ,  $i = 0, \dots, N - 1$ . For simplicity, we assume  $N = 2^m$ . So each ID is represented with  $m$  bits. Inspired by [12], we consider that the nodes are arranged in a circle in the order of their ID, which increases in a clockwise direction. The *distance* between two nodes is defined as the length of the shorter path along the circle from one node to the other. The shorter path can be either clockwise or counterclockwise. Let  $D(i, j)$  denote the distance between nodes  $i$  and  $j$ . It can be computed as  $D(i, j) = \min(i - j, j - i)^1$ . When preloading keys, we let node  $i$  preshare a key with nodes  $i \pm 2^0, i \pm 2^1, \dots, i \pm 2^{m-2}, i \pm 2^{m-1}$ . That is, nodes  $i$  and  $j$  preshare a key if and only if the distance between them is a power of two, i.e.,  $D(i, j) = 2^x$  for some  $x \in \{0, \dots, m - 1\}$ . Let  $K_{i,j}$  denote the key preshared by nodes  $i$  and  $j$ . Then  $K_{i,j} = K_{j,i}$ . On the circle, we connect two nodes with a chord if they preshare a key. During key establishment, a session key is forwarded along the chords from a source to a destination. So we name our scheme a *Chord-based Key Establishment* (CBKE) scheme.

With the above key pre-distribution, each node shares a unique key with  $2m - 1 = 2 \log_2 N - 1$  nodes. The number of preloaded keys on each node can be reduced to  $m + 1$ , using the method proposed in [6].

Fig. 1 shows an example of key pre-distribution for a network of 16 nodes, in which node 0 preshares keys with nodes 1, 2, 4, 8, 12, 14 and 15. In the figure, node 0 is connected with each of these nodes with a chord.

<sup>1</sup>All computations involving node IDs are modulo  $N$ . For simplicity, we do not indicate it explicitly.

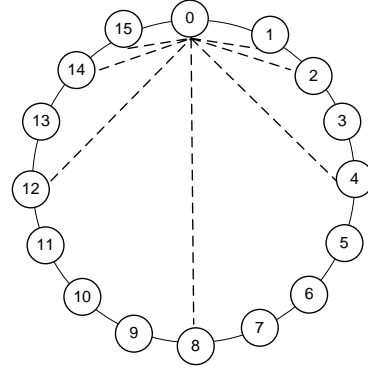


Fig. 1. An illustration of key pre-distribution in CBKE ( $N = 16$ ).

**2.1.2. Key establishment in basic CBKE:** Suppose that a source node  $s$  wants to establish a session key with a destination node  $d$ . For this purpose,  $s$  first generates a random key  $r$ . The key establishment process is basically forwarding the key to  $d$  through a secure path.  $s$  confirms the success of key establishment if it receives an ACK message from  $d$  that is encrypted with  $r$ . Let  $k = D(s, d)$ ,  $k_i$  denote bit  $i$  in the binary representation of  $k$ , and  $H(k)$  be the *Hamming weight* of  $k$  (i.e., the number of 1's in  $k$ 's binary representation). Since nodes  $s$  and  $d$  are not the same,  $k \neq 0$  and  $H(k) > 0$ . We now describe the key establishment process, as summarized in Fig. 2.

Node  $s$  finds a node  $n_1$  that is closer to  $d$  and preshares a key with  $s$ . In particular,  $s$  randomly selects  $u$  such that  $k_u = 1$  and computes  $n_1 = s \pm 2^u$ . The operation,  $+$  or  $-$ , depends on whether the shorter path from  $s$  to  $d$  is clockwise or counterclockwise. If it is clockwise,  $+$  is performed. Otherwise,  $-$  is performed. Since  $k$  has  $H(k)$  1 bits and  $s$  may select any of these 1's,  $s$  have  $H(k)$  different choices for  $n_1$ . We refer to these nodes, which can be selected by  $s$  to forward the session key, as *descendants* of  $s$ . It is clear that  $n_1$  is closer to  $d$  than  $s$  because  $D(n_1, d) = D(s, d) - 2^u < D(s, d)$ . After  $n_1$  is selected,  $s$  sends  $(r, d)$  to  $n_1$ . The message is encrypted with  $K_{s,n_1}$ , which is the key preshared by  $s$  and  $n_1$ .

After  $n_1$  receives the encrypted message from  $s$ , it recovers  $r$  and  $d$  with  $K_{s,n_1}$ . If  $n_1$  is not the destination node, it selects one of its descendants  $n_2$  and forwards  $(r, d)$  to it. If  $n_2$  is not  $d$  either, it will repeat the process. Since  $r$  gets closer to  $d$  after each hop, it will eventually arrive at  $d$ .

Once  $d$  receives  $r$ , it sends an ACK message to  $s$ . The ACK message can be  $(s, d, \rho)$  encrypted with  $r$ , where  $\rho$  is a random value generated by  $d$ . If  $s$  can retrieve  $s$  and  $d$  with  $r$  from the ACK message, it assumes that  $d$  has received  $r$  successfully. Then it can start a secure communication session with  $d$ .

We refer to a path along which a session key is

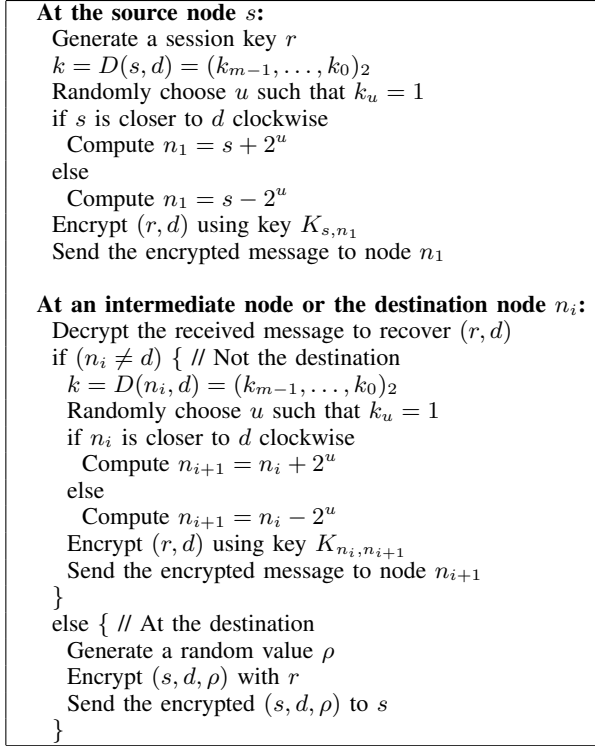


Fig. 2. Basic CBKE scheme: node  $s$  establishes a session key with node  $d$ .

forwarded securely from  $s$  to  $d$  as a *logical path*. On a logical path, a node and its descendant form a *logical hop* since they are not necessarily neighboring nodes in the physical network. Two nodes on a logical hop preshare a key, which is used to protect key forwarding messages. A logical path is only used for key forwarding — the ACK message is sent directly from  $d$  to  $s$ ; after the session key is established, the secure communications between  $s$  and  $d$  also take a direct path.

We illustrate the basic scheme with an example shown in Fig. 3. Suppose  $N = 256$  and node 0 wants to send a session key  $r$  to node 83. The distance between the two nodes is  $k = D(0, 83) = 83 = 01010011_2$ . Since  $H(k) = 4$ , node 0 has four descendants: nodes  $64 = 0 + 2^6$ ,  $16 = 0 + 2^4$ ,  $2 = 0 + 2^1$ , and  $1 = 0 + 2^0$ , where the exponents 6, 4, 1, and 0 are the position of 1's in  $k$ . Node 0 randomly picks one of the descendants, say node 64, and forwards  $r$  and the destination ID 83 to it. The message is protected with key  $K_{0,64}$ . After receiving and decrypting the message, node 64 finds out that it is not the destination. It then forwards the message to one of its descendants (nodes 65, 66, and 80). This process continues until node 83 receives  $r$ .

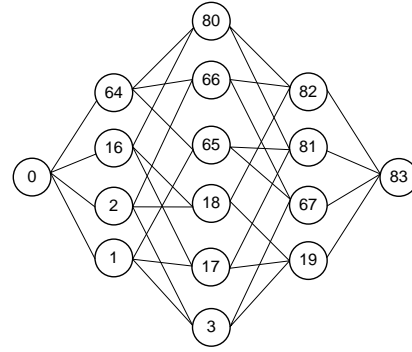


Fig. 3. Logical paths from source 0 to destination 83.

**2.1.3. Utilizing multiple logical paths:** In the basic CBKE scheme shown in Fig. 2, a session key is forwarded along a single logical path from  $s$  to  $d$ . A single path, however, does not provide sufficient reliability for key establishment if nodes may fail. As we see in Fig. 3, many logical paths can be utilized to forward a key from  $s$  to  $d$ . Forwarding a session key simultaneously through multiple logical paths improves reliability. The key can reach the destination node even if all but one logical path are broken.

We introduce a parameter  $l$  to our basic scheme and allow a node to forward a session key to  $l$  descendants simultaneously. In particular, a node  $n_i$  forwards the session key to  $\min(l, H(D(n_i, d)))$  descendants simultaneously. The message to each of the  $l$  descendants is protected by the unique key preshared by  $n_i$  and its descendants. A descendant may receive the same session key multiple times due to the overlaps among logical paths, but it only forwards the key once.

When  $l = 1$ , only a single logical path is used to forward the session key from  $s$  to  $d$ , as illustrated in Fig. 2. When not restricting  $l$ , i.e., a node forwards the key to all its descendants, the CBKE scheme works in the *flooding* mode. There is a clear trade-off in choosing  $l$ : a large  $l$  improves the reliability of key establishment while increasing communication cost. We will explore the trade-offs in Section 4.

We again use the example in Fig. 3 to illustrate forwarding along multiple paths. Suppose  $l = 2$ . Each node forwards the session key to up to two descendants. Source node 0 randomly chooses two nodes from its four descendants, nodes 64, 16, 2, and 1. Suppose nodes 64 and 16 are selected. Node 0 forwards the session key to these two nodes simultaneously. The message to node 64 is encrypted with  $K_{0,64}$  and the message to node 16 is encrypted with  $K_{0,16}$ . After receiving the (encrypted) session key, node 64 chooses two descendants from nodes 65, 66 and 80. Node 16 also chooses two

of its descendants randomly. Note that node 80 is a descendant for both nodes 64 and 16. It may receive the session key twice, but it only forwards the key once. Since the session key is forwarded along multiple paths, it can reach the destination node 83 even if some nodes (e.g., node 16) fail.

To reduce communication cost, multiple logical paths can be utilized in a different way. They do not have to be involved in forwarding simultaneously. Instead they can be utilized in series or in a combination of parallel and series. The source node  $s$  can keep trying to forward a session key to  $d$  with a certain  $l$  that incurs less communication cost (e.g.  $l = 1$  or  $l = 2$ ) until the key is successfully sent or a predetermined number of trials is reached. When  $l = 1$ , the multiple logical paths are tried in series. Since each time the descendants involved in forwarding are randomly chosen, it is very likely that different logical paths are taken at different times. This method can achieve similar reliability as trying multiple logical paths simultaneously with a larger  $l$ . However, it requires other mechanisms such as ACK messages or time-out to detect failed attempts.

## 2.2. Generalized CBKE scheme

In the basic CBKE scheme, we consider  $k$  as a binary number and change a 1 bit to 0 after each logical hop when forwarding the session key from  $s$  to  $d$ . We can generalize the scheme by representing  $k$  as a radix  $2^b$  number, where  $b \geq 1$ , and changing a non-zero digit of  $k$  to 0 when forwarding the session key to  $d$ . In the generalized CBKE scheme,  $H(k)$  represents the number of non-zero digits in  $k$ .

**Key pre-distribution.** When the radix is  $2^b$ ,  $k$  can be represented with  $g$  digits  $(k_{g-1}, k_{g-2}, \dots, k_1, k_0)$ , where  $g = \lceil m/b \rceil$ . Each digit  $k_i$ ,  $i = 0, \dots, g-1$ , has  $2^b$  different values. Two nodes pre-share a key if their distance is  $a \times 2^{ib}$ , where  $0 < a < 2^b$  and  $0 \leq i < g$ . In other words, if the distance between two nodes has only one non-zero digit, they pre-share a key.

Because  $k$  has  $g$  digits and each digit has  $2^b - 1$  non-zero values, a node needs to establish a unique key with  $2 \times (2^b - 1) \times (g - 1) + (2^b - 1)$  nodes. Using the method proposed in [6], each node needs to store  $(2^b - 1) \times g + 1 = (2^b - 1) \times \lceil m/b \rceil + 1$  keys.

We use an example to illustrate key pre-distribution in the generalized CBKE scheme. Suppose  $N = 256$  and  $b = 2$ . The distance between two nodes can be represented with a 4-digit radix-4 number. Node 0 pre-shares a unique key with 21 nodes, which are listed in Table 1. In the table, we use a subscript 4 to indicate radix-4 numbers. We also group the nodes according to the position of the non-zero

digit in their distance to node 0. In general, the nodes pre-sharing a key with a particular node can be divided into  $g$  groups as  $k$  has  $g$  digits.

Table 1. Nodes pre-sharing a key with node 0 ( $N = 256, b = 2$ ).

Distance to node 0	Node IDs	Group
0001 <sub>4</sub> = 1	1 and 255	1
0002 <sub>4</sub> = 2	2 and 254	
0003 <sub>4</sub> = 3	3 and 253	
0010 <sub>4</sub> = 4	4 and 252	2
0020 <sub>4</sub> = 8	8 and 248	
0030 <sub>4</sub> = 12	12 and 244	
0100 <sub>4</sub> = 16	16 and 240	3
0200 <sub>4</sub> = 32	32 and 224	
0300 <sub>4</sub> = 48	48 and 208	
1000 <sub>4</sub> = 64	64 and 192	4
2000 <sub>4</sub> = 128	128	

**Key establishment.** The key establishment process in the generalized CBKE is similar to the process in the basic CBKE, which is shown in Fig. 2. The only difference is that at each hop, a node  $n_i$  randomly selects a non-zero digit  $k_u$  in  $k = D(n_i, d)$  and compute  $n_{i+1} = n_i \pm k_u \times 2^{bu}$ . Similarly,  $+$  is performed if  $n_i$  is closer to  $d$  clockwise. Otherwise,  $-$  is performed. After each logical hop, one of the non-zero digits in  $k$  is changed to 0. Consequently, a logical path has at most  $g$  hops. On average, the larger  $b$  is, the shorter logical paths we have.

As in the basic CBKE scheme, we can use  $l$  to control the number of descendants to which a node may forward a session key. Also, the multiple paths can be utilized in parallel, in series, or in a combination of both.

We use the example in Fig. 4 to illustrate the generalized CBKE scheme. Suppose  $N = 256$ ,  $b = 2$ ,  $l = 1$ , and node 0 wants to send a session key  $r$  to node 83. The distance between the two nodes is  $k = D(0, 83) = 83 = 1103_4$ . Since  $k$  has three non-zero digits,  $H(k) = 3$ . Node 0 has three descendants: nodes 3, 16, and 64. It randomly picks one of the descendants, say node 3, as the next node on the logical path. After node 3 receives  $r$ , it selects either node 67 or node 19 as the next node, which then forwards  $r$  to node 83. Because of the larger radix, a logical path now has only three hops, instead of four hops in the case shown in Fig. 3.

**Relationship to existing deterministic schemes.** We remark that several existing deterministic schemes are special cases of CBKE. The full pairwise scheme is CBKE with  $b = m$ , where each pair of nodes pre-share a unique key. PIKE [1] is essentially a CBKE scheme with  $b = m/2$ , where each node pre-stores  $O(\sqrt{N})$  keys and each logical path contains two logical hops. When nodes in a group share the same key, CBKE with  $b = m$  has a single group and all nodes share the same key, which

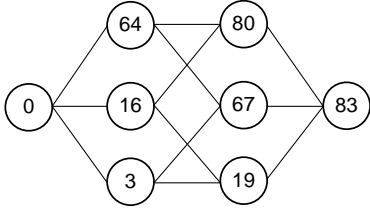


Fig. 4. Logical paths from source 0 to destination 83 ( $b = 2$ ).

becomes the single-key predistribution scheme.

### 3. Properties of CBKE Schemes

We next describe several properties of the CBKE schemes. Recall that  $k = D(s, d)$ , where  $s$  is the source node and  $d$  is the destination node. We consider  $k$  as a radix  $2^b$  number and  $H(k)$  is the number of non-zero digits in  $k$ .

**Memory overhead.** As mentioned earlier, the number of preloaded keys into each node is  $(2^b - 1) \times \lceil m/b \rceil + 1$ , where  $m$  is the number of bits required to represent a node ID. Hence the memory overhead increases with  $b$ .

**Length of a logical path.** When the session key is forwarded from  $s$  to  $d$ , the number of non-zero digits in  $k$  is reduced by one after each logical hop. Eventually the distance becomes 0 at  $d$ . Therefore, a logical path from  $s$  to  $d$  has  $H(k)$  logical hops. With a larger radix,  $k$  has fewer digits. On average,  $H(k)$  is reduced and so is the length of logical paths.

**Number of logical paths and hops.** There are more than one logical path from  $s$  to  $d$  if  $H(k) > 1$ . We now compute the total number of logical paths that can be used for key establishment. Let  $S$  denote the set of nodes that may participate in forwarding and  $S_i$  denote the set of nodes in  $S$  whose distance to  $s$  has  $i$  non-zero digits,  $i = 0, \dots, H(k)$ . In particular,  $S_0 = \{s\}$ ,  $S_{H(k)} = \{d\}$ . Starting from  $S_0$ , the session key is forwarded to a node in  $S_1$ , and then to a node in  $S_2$ , and so on. Consider a node  $n_i \in S_i$ . We have  $D(n_i, d) = H(k) - i$ . Therefore, node  $n_i$  has  $H(k) - i$  descendants, all of which are in  $S_{i+1}$ . Node  $n_i$  has  $(H(k) - i)$  choices when selecting a descendant in  $S_{i+1}$ . For example,  $s$  has  $H(k)$  choices. And a node in  $S_1$  has  $H(k) - 1$  choices. Hence, the total number of logical paths from  $s$  to  $d$  is

$$\prod_{i=0}^{H(k)-1} (H(k) - i) = H(k)!.$$

The number of nodes in set  $S_i$  is  $\binom{H(k)}{i}$  because  $i$  out of  $H(k)$  non-zero digits in  $k$  have changed to 0 for a node in  $S_i$ . Combining with the previous observation that each node in  $S_i$  has  $H(k) - i$  different ways to reach nodes in  $S_{i+1}$ , we obtain the total number of logical hops that may be used in forwarding as

$$\sum_{i=0}^{H(k)-1} (H(k) - i) \binom{H(k)}{i} = H(k) 2^{H(k)-1}.$$

We now look at some examples. In Fig. 3,  $b = 1$ ,  $S_0 = \{0\}$ ,  $S_1 = \{1, 2, 16, 64\}$ ,  $S_2 = \{3, 17, 18, 65, 66, 80\}$ ,  $S_3 = \{19, 67, 81, 82\}$ , and  $S_4 = \{83\}$ . Each logical path from node 0 to node 83 contains  $H(k) = H(01010011_2) = 4$  logical hops. The total number of logical paths from node 0 to node 83 is  $H(k)! = 4! = 24$ . The total number of logical hops is  $H(k) 2^{H(k)-1} = 4 \times 2^3 = 32$ . In Fig. 4,  $b = 2$ ,  $S_0 = \{0\}$ ,  $S_1 = \{3, 16, 64\}$ ,  $S_2 = \{19, 67, 80\}$ , and  $S_3 = \{83\}$ . Each logical path from node 0 to node 83 has  $H(k) = H(1103_4) = 3$  logical hops. The total number of logical paths from node 0 to node 83 is  $H(k)! = 3! = 6$ . The total number of logical hops is  $H(k) 2^{H(k)-1} = 3 \times 2^2 = 12$ .

**Summary.** In summary, when  $b$  increases, the memory overhead increases, while, on average, the length of a logical path and the total number of logical paths decrease. If we assume that nodes in a network are uniformly distributed and node failures are also uniformly random, then each logical hop contains a similar number of physical hops and has a similar probability to break. Therefore, on average, a shorter logical path incurs less communication cost and provides better reliability. However, when a logical path is short, the total number of logical paths from a source to a destination is also small, which limits the number of logical paths that can be used simultaneously for key establishment, and hence limits the degree of reliability that can be achieved. In Section 4, we will study how the choice of  $b$  affects communication cost and reliability in detail.

### 4. Performance Trade-offs

In this section, we explore how the choices of  $b$  and  $l$  affect the *memory overhead*, *reliability*, and *communication cost* of CBKE. The memory overhead is the number of keys that are preloaded into a sensor node. The reliability is the probability that a pair of nodes can establish a session key successfully when nodes may fail. The communication cost is the average number of messages transmitted in the network for a key establishment. We developed our own simulator for fast simulations.

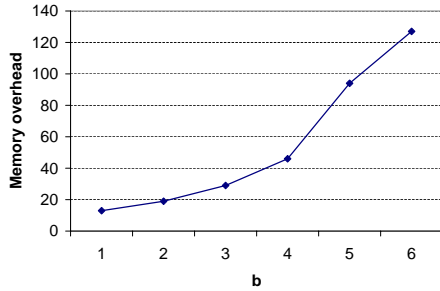


Fig. 5. Memory overhead.

We assume a network of  $N = 4096$  nodes ( $m=12$ ) and each node has a fixed physical position. The transmission range of a node is one unit and the node density is 20 nodes per square unit. We adopt GPSR [7] as the routing protocol in the physical network. So every node knows the position of the destination node and its neighbouring nodes. We set  $b$  and  $l$  to a wide range of settings:  $b = 1, 2, \dots, 6$ , and  $l = 1, 2, 3$  and unlimited (i.e., in the flooding mode). Note that when  $b = 6$ , CBKE becomes PIKE [1].

We first compare the memory overhead for different choices of  $b$ . As we discussed in Section 2.2, the number of preshared keys on each node is  $(2^b - 1) \times \lceil m/b \rceil + 1$ , which increases as  $b$ . In particular, when  $b = 1$ , the number of preshared keys is  $O(\log_2 N)$ ; when  $b = m/2$ , the number of preshared keys is  $O(\sqrt{N})$ . Fig. 5 plots the memory overhead for various values of  $b$ . We observe that, as  $b$  increases from 1 to 6, the number of preloaded keys first increases gradually and then increases sharply. Therefore, when memory is a stringent constraint, it is desirable to choose smaller values of  $b$ .

We now explore how the values of  $b$  and  $l$  affect the reliability and communication cost of CBKE. We set node failure rate,  $p$ , to values between 0.1 and 0.5, with an increment of 0.1. For each value of  $p$ , we randomly choose 10,000 pairs of alive nodes in the network and establish keys for each pair. We then obtain the average reliability and communication cost over all the pairs.

Fig. 6(a) and (b) plot the average reliability and communication cost for  $p = 0.3$ . We observe that when  $l = 1$ , the average reliability increases significantly with  $b$ . This is because larger values of  $b$  lead to shorter logical paths and thus to higher probabilities of successful key establishments. As  $l$  increases from 1 to 2, the average reliability for all values of  $b$  increases significantly, indicating the effectiveness of using multiple paths to increase reliability. When  $l = 3$ , we observe that the average reliability decreases with  $b$ . This is because when  $l = 3$ , more logical paths can be utilized for key establishment for small values of  $b$  and hence result in

good reliability. As  $b$  increases, the number of logical paths that can be utilized for key establishment decreases significantly. Although each logical path is more likely to be successful, the overall reliability decreases because of the small number of logical paths. When not limiting  $l$ , we observe a similar trend as when  $l = 3$ . In Fig. 6(b), as expected, we observe that the average communication cost decreases with  $b$  since the average length of a logical path decreases as  $b$  increases. Furthermore, as  $l$  increases, the average communication cost may increase significantly, especially when  $b$  is small, since a much larger number of logical hops are involved.

We observe similar trends in reliability and communication cost for other values of  $p$  (0.1, 0.2, 0.4, and 0.5). In general, the average reliability increases with  $b$  for small values of  $l$ , and decreases with  $b$  for large values of  $l$ ; the highest reliability is achieved when  $b = 1$  and not limiting  $l$ , at the price of the highest communication cost.

In summary, we see a clear trade-off in memory overhead, reliability, and communication cost when choosing different values of  $b$  and  $l$  in CBKE. The best choices of the parameters depends on the constraints of a specific setting. For instance, one may determine  $b$  and  $l$  to achieve the highest reliability for certain constraints of memory overhead and communication cost. By tuning the parameters of  $b$  and  $l$ , CBKE provides a wide range of performance trade-offs among memory overhead, reliability, and communication cost.

## 5. Further Improvements to CBKE

CBKE can be further improved in several ways. Due to the space limit, we only briefly discuss a few of them here.

**Resilience to compromise.** In CBKE, a logical path from  $s$  to  $d$  has  $H(k)$  hops, where  $H(k)$  is the number of non-zero digits in  $k$ . Assume the probability of a node being compromised is  $p$ . Then the probability of the session key being compromised is  $1 - (1-p)^{H(k)}$ . As  $H(k)$  increases, the probability that a session key is compromised increases. To improve security, one can use a larger radix to reduce the number of hops on a logical path, as discussed in Section 2.2. Alternatively,  $s$  can send several secret values to  $d$  over different paths. One needs to know all the values to construct the session key. For example,  $s$  may send two secret values to  $d$ , one through the clockwise path and the other through the counterclockwise path. The destination node  $d$  can construct the session key by exclusive or-ing the two values. Even if an adversary compromises one or more nodes on one of the paths, he cannot construct the session key. However, sending

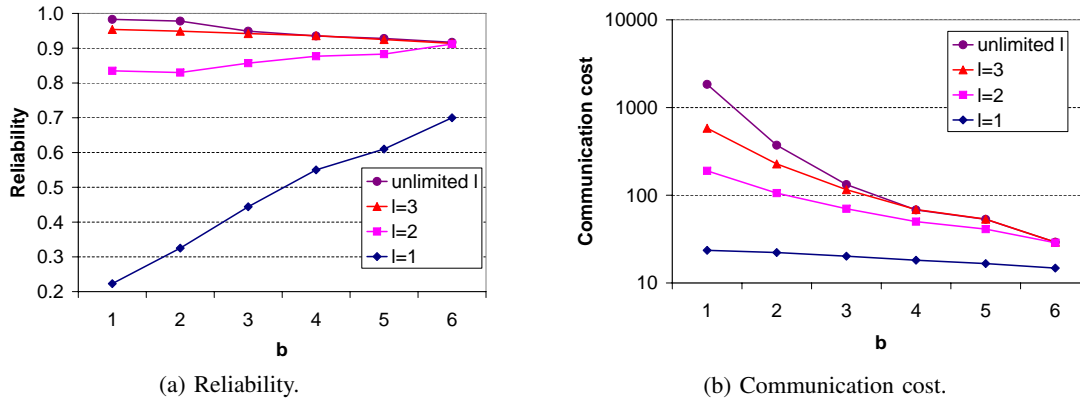


Fig. 6. Reliability and communication cost,  $p = 0.3$ .

multiple values increases communication cost. The trade-off between security and communication cost is left as future work.

**Choice of forwarding paths.** In CBKE, the total number of logical paths from  $s$  to  $d$  is  $H(k)!$ . For relatively small  $H(k)$  (e.g.,  $H(k) = 1$  or  $2$ ),  $H(k)!$  logical paths may not provide sufficient amount of path redundancy. More logical paths can be utilized if one chooses additional intermediate nodes. For instance, suppose  $s = 0$ ,  $d = 6$ , and  $b = 1$ . The current CBKE scheme provides only two paths:  $(0, 2, 6)$  or  $(0, 4, 6)$ . If both nodes 2 and 4 fail, the session key cannot be forwarded to node 6. However, it can be done through additional paths such as  $(0, 8, 6)$  and  $(0, 1, 5, 6)$ . The additional forwarding paths may also reduce the length of logical paths. For example, when  $b = 1$ , a logical path between nodes 0 and 7 has three hops in the current scheme. A shorter path  $(0, 8, 7)$  can be utilized to reduce communication cost.

**Partial deployment.** So far, we have assumed that the number of nodes in the network is a power of two. In a real deployment, the number of nodes in a network may not be a power of two. A simple way to deal with this problem is to consider undeployed nodes as failed nodes. A more advanced deployment and forwarding scheme can reduce communication overhead.

## 6. Conclusions

In this paper, we propose CBKE, a pairwise key establishment protocol for large-scale sensor networks. Compared to previously proposed schemes such as PIKE, our scheme provides more flexibility to balance memory overhead, reliability, and communication cost. We also

evaluated the trade-offs using simulation. In the future, we plan to enhance CBKE in several ways such as improving its resilience to compromise and achieving better reliability with low communication overhead.

## References

- [1] H. Chan and A. Perrig. PIKE: Peer intermediaries for key establishment in sensor networks. In *IEEE INFOCOM*, 2005.
- [2] W. Du, J. Deng, Y. S. Han, S. Chen, and P. Varshney. A key management scheme for wireless sensor networks using deployment knowledge. In *Proc. IEEE INFOCOM*, Hong Kong, March 2004.
- [3] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *CCS*, pages 42–51, New York, NY, 2003.
- [4] L. Eschenauer and V. Gligor. Key management scheme for distributed sensor networks. In *CCS*, pages 41–47, 2002.
- [5] A. Perrig, H. Chan and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, pages 197–213, 2003.
- [6] S. I. Huang. Adaptive random key distribution schemes for wireless sensor networks. In *Proceedings of the International Workshop on Advanced Developments in Software and Systems Security*, 2003.
- [7] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *MobiCom*, pages 243–254, 2000.
- [8] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *CCS*, New York, NY, 2003.
- [9] D. Liu, P. Ning, and W. Du. Group based key pre-distribution in wireless sensor networks. In *Wise*, 2005.
- [10] F. Liu and X. Cheng. A self-configured key establishment scheme for large-scale sensor networks. In *MASS*, 2006.
- [11] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *MobiCom*, pages 189–199, 2001.
- [12] I. Stoica, R. Morris, D. L. Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *SIGCOMM*, pages 149–160, San Diego, CA, 2001.
- [13] W. Zhang, M. Tran, S. Zhu, and G. Cao. A random perturbation-based scheme for pairwise key establishment in sensor networks. In *ACM MobiHoc*, Montreal, QC, Canada, September 2007.
- [14] L. Zhou, J. Ni, and C. V. Ravishanker. Efficient key establishment for group-based wireless sensor networks. In *Wise*, 2005.