SMOCK: A Self-contained Public Key Management Scheme for Mission-critical Wireless Ad Hoc Networks

Wenbo He Ying Huang Klara Nahrstedt
Department of Computer Science
201 N. Goodwin Avenue
Urbana, IL 61801-2302

Whay C. Lee
Motorola Labs
111 Locke Drive
Marlborough, MA 01752

Abstract

Mobile ad hoc networks show great potential in emergency response and/or recovery. Such mission-critical applications demand security service be "anywhere", "anytime" and "anyhow". However, it is challenging to design a key management scheme in current wireless ad hoc networks to fulfill the required attributes of secure communications, such as data integrity, authentication, confidentiality, non-repudiation *and* service availability, *when* Sybil *attacks are present. In this paper, we present a self-contained public key management scheme, called* SMOCK, *which is able to resist the* Sybil *attack, achieves zero communication overhead for authentication, and offers high service availability. In our scheme, small number of cryptographic keys are stored off-line at individual nodes before they are deployed in the network. To provide good scalability in terms of number of nodes and storage space, we utilize a combinatorial design of public-private key pairs, which means nodes combine more than one key pair to encrypt and decrypt messages. We also show that* SMOCK *provides controllable resilience when malicious nodes break into a limited number of nodes before key revocation and renewal.*

1 Introduction

There are emerging needs of secure communications in mission-critical applications over wireless ad hoc networks, including battlefield communications, emergency rescue operations, and disaster recoveries. In these applications, it is important to support secure communications in "anywhere", "anytime" and "anyhow" manner with following attributes [1] [3]: *data integrity*, *authentication*, *confidentiality*, *non-repudiation*, and *service availability*. Being good candidates to address these attributes, Public-Key Cryptography (PKC) schemes have advantages over the symmetric systems [2]. However, characteristics of mission-critical ad hoc networks pose the following new challenges for the design of public key management schemes that would support secure communication over wireless ad hoc networks:

(1) Vulnerability to the *Sybil* Attack: Wireless communications are prone to both active and passive attacks. The *Sybil* attack [4] is an active attack and particularly detrimental to mobile ad hoc networks. When the *Sybil* attack happens, an attacker can claim multiple identities and the fake identities can easily defeat reputation and threshold protocols, where a legitimate node must rely on majority of nodes to reach decisions. Therefore, a node should not trust others unless the node can infer someone else is trustable from local information.

(2) Unreliable Communications and Network Dynamics: Due to shared-medium nature of wireless links, flows may frequently interfere with each other. Moreover, a network may be partitioned frequently due to node mobility and poor channel condition. Mobile nodes may leave and join the ad hoc network frequently and new legitimated nodes may join the network later after some nodes have been deployed in the field. Mobility increases the complexity for trust management.

(3) Large Scale: The number of ad hoc wireless devices deployed at an incident scene depends on specific nature of the incident. In general, the network size can be very large. In addition, an ad hoc network should be able to accommodate more mobile devices if necessary, therefore it is necessary to have newly deployed devices and previously deployed devices trust each other without introducing too much overhead.

(4) Resource Constraints: The wireless devices usually have limited bandwidth, memory and processing power. Among these constrains, communication bandwidth consumption and memory are two big concerns for key management schemes. Wireless bandwidth is the scarcest re-

sources in wireless network. On the other hand, memory concern for key storage is more and more evident, since the requirement on network scalability (or network size) is increasing.

Given the above challenges (1) and (2), a node in a network may encounter untrustworthy peers and unreliable communication. Therefore, we need a self-contained key management scheme. A realistic assumption about mission-critical applications is that: Before mobile devices are dispatched to an incident area, they are able to communicate securely with the trusted authentication server in their domain center, and get prepared before their deployment. Once the wireless devices are dispatched into the incident area, the centralized trusted server loses control of these devices and the mobile devices cannot trust anybody if local information cannot authenticate it. In this paper, we design a self-contained public-key management scheme, where all necessary cryptographic keys (certificates) are stored at individual nodes before nodes are deployed in the incident area. As a result, we can expect almost zero communication overhead for authentication. In contrast to traditional certificate-based schemes, our authentication procedure does not require the communication of certificate, binding the node's ID to its public key, and signed by an off-line trusted authority. The required storage space for traditional self-contained public key management schemes is of $O(n)$ order. With challenges (3) and (4), storage space at individual nodes may be too small to accommodate self-contained security service, when network size n is large. Hence, we present a Scalable Method Of Cryptographic Key (*SMOCK*) management scheme, which scales logarithmically with network size, $O(log\, n)$, with respect to storage space.

In order for *SMOCK* to use smaller set of cryptographic keys, a sender uses multiple keys to encrypt a message and a receiver needs multiple keys to decrypt the message. We then use the public key cryptography as follows: Each node possesses a unique combination of private keys, and knows all public keys. The private key combination pattern is unambiguously associated with the node ID. It means, if a sender A wants to send a message to receiver B, A will first acquire B's ID to infer a set of private keys owned by B. Then A will encrypt the message with the public key set that corresponds to the private keys owned by B. We have evaluated *SMOCK* with respect to the communication overhead for key management, memory footprint, and resilience to node break-in by adversaries. Note that it is likely that adversaries may eventually break into a limited number of nodes over a certain period before a network detects the break-in and revokes the compromised keys. However, before the system detects break-ins, a majority of network nodes under the *SMOCK* will operate securely even when a small amount of nodes are compromised.

The paper is organized as follows: In Section 2, we summarize the related work in key management schemes in mobile ad hoc networks. In Section 3, we describe the background and problem description. Section 4 provides the details of our key allocation algorithms. Section 5 gives detailed protocols for secure communication and bootstrapping when new nodes are deployed. Section 6 evaluates the proposed scheme. Finally, Section 7 provides concluding remarks.

2 Related Work

For secure communication, wireless sensor networks use symmetric key techniques [11, 12, 13, 14, 15, 16, 17]. The main advantage of symmetric key techniques is its computational and energy efficiency. In symmetric key techniques, secret keys are pre-distributed among nodes before their deployment. A challenge of the key distribution scheme is to use small memory size to establish secure communication among a large number of nodes and achieve good resilience. In sensor network context, the "security" emphasizes link layer security, and the major security goal is to prevent outsiders (adversaries) to use network resources. Due to the lack of support for *authentication* and *confidentiality*, [12] and [13] are not suitable in mission-critical applications over wireless ad hoc networks. Pairwise key distribution schemes [14] [16] are able to bolster *authentication*. However, the connectivity is still probabilistic in these schemes and there could be some partitions in the network. To fully support required features of mission-critical networks, including *data integrity*, *authentication*, *confidentiality*, *non-repudiation* and *service availability*, we consider public key schemes for secure communication over wireless ad hoc networks in this paper.

Public key (certificate) based approaches were originally proposed to provide solutions to secure communications for the Internet [5], where security services rely on a centralized certification server. However, with a centralized server, security service for mission-critical applications may suffer from low availability and poor scalability due to the low reliability and poor connectivity of mobile ad hoc networks. Also, a single point failure of centralized server is able to paralyze the whole network, which makes the network extremely vulnerable to compromises and denial-of-service attacks. To improve resilience to break-ins in wireless ad hoc networks, Zhou and Haas tailor the certificate-based approaches to ad hoc networks and present a distributed public-key management scheme for ad hoc networks [3], where multiple distributed certificate authorities are used. To sign a certificate, each authority generates a partial signature for the certificate and submits the partial signature to a coordinator that calculates the signature from the partial signatures. Kong et al. describe a similar but fully distributed scheme [6], where every node carries a share of the private key. In certification service, if a node collects K partial certificates from its one-hop neighbors, the node is able to obtain its complete certificate. This scheme increases availability and reduces multi-hop communication of authentication service. However, such a system is vulnerable to the Sybil attack [4], where an attacker can claim multiple identities (larger than K), and cheats honest nodes with the fake partial certificate. To improve security service availability and system scalability, Capkun, Buttyan, and Hubaux propose a self-organized public key management system [7], where users issue certificates based on their personal acquaintances. Each user maintains a local certificate repository. When two users want to verify the public keys of each other, they merge their local certificate repositories and try to find (within the merged repository) appropriate certificate chains that make the verification possible. However, it yields low security assurance when *Sybil* attacks are present due to the lack of trust anchor, since *Sybil* attackers can easily defeat reputation and threshold protocols [6] [7] [8] [9] [10]. Therefore, we turn to self-contained public key approaches for high security assurance and low communication overhead.

Recent development of ECC algorithms and implementations [18] [19] [20] show that ECC is an efficient PKC scheme and it is becoming very feasible to consider ECC based PKC schemes in mobile ad hoc networks. Currently, there are ongoing efforts to include ECC as a recommended security mechanism, such as IEEE 802.15 WPAN, OMA (Open Mobile Alliance), and IETF: IPSec, TLS, PKIX, S/MIME. To bridge the gap between the development of PKC technologies and the use of PKC in mobile ad hoc networks, scalable key management schemes are needed.

3 Problem Statement

In SMOCK, let us assume a group of people in an incident area, who want to exchange correspondence securely among each other in a pair-wise fashion[1]. The key pool $\mathcal{K}$ of such a group consists of a set of private-public key pairs, and is maintained by an off-line trusted server. Each key pair consists of two mathematically related keys. The i-th key pair in the key pool is represented by (k^i_{priv}, k^i_{pub}). To support secure communication in the group, each member is loaded with all public keys of the group and assigned a distinct subset of private keys. Let $\mathcal{K}^{priv}_{Alice}$ denote a subset of private keys held by Alice, and $\mathcal{K}^{pub}_{Alice}$ represents Alice's corresponding public key subset. If Bob wants to send a secret message to Alice, he needs to know $\mathcal{K}^{pub}_{Alice}$, where $\mathcal{K}^{priv}_{Alice} \not\subset \mathcal{K}^{priv}_{anybody\_else}$. Bob is able to pass the secret message to Alice, using the public keys $\mathcal{K}^{pub}_{Alice}$ to encrypt the message. The message can be opened only by Alice, who has the private key set $\mathcal{K}^{priv}_{Alice}$, but others do not.

[1]Symbols and terms used throughout this paper are shown as in Table 1

Table 1. Notations and Symbols

$\mathcal{K}$	A Key pool: a set of public-private key pairs
$privateKey_{ij}$	j-th private key hold by user i
$publicKey_{ij}$	j-th public key hold by user i
$\mathcal{K}^{priv}_i$	A set of private keys held by user i, $\mathcal{K}^{priv}_i = \{privateKey_{ij}\}$
$\mathcal{K}^{pub}_i$	A set of public keys corresponding to $\mathcal{K}^{priv}_i$
$\mathcal{K}_i$	A set of public-private key pairs held by user i, $\mathcal{K}_i = \{(k_{priv}, k_{pub}) \mid k_{priv} \in \mathcal{K}^{priv}_i \; \& \; corresponding \; k_{pub} \in \mathcal{K}^{pub}_i\}$
M	Memory size for key storage
a	Number of distinct key pairs $a = \lvert\mathcal{K}\rvert$
b	Number of private keys held by each user under isometric key allocation, $b = \lvert\mathcal{K}_1\rvert = \lvert\mathcal{K}_2\rvert = \cdots = \lvert\mathcal{K}_n\rvert$
$k_c(x)$	Expected number of disclosed keys when x nodes are broken in
$k_v(x)$	Maximum number of disclosed keys when x nodes are broken in
$V_x(a,b)$	Vulnerability metrics as x nodes are broken in.
$C(a,b)$	Abbreviation of a choose b, $\binom{a}{b}$
V	A set of nodes in the ad hoc wireless network
n	Total number of nodes in the network, $n = \lvert V\rvert$

Consider an example of a small group with 10 users. In *SMOCK*, we need 5 distinct public-private key pairs to build pair-wise secure communication channels among 10 users. They are (k^1_{priv}, k^1_{pub}), (k^2_{priv}, k^2_{pub}), (k^3_{priv}, k^3_{pub}), (k^4_{priv}, k^4_{pub}), (k^5_{priv}, k^5_{pub}). Each user keeps 5 public keys and 2 private keys. The unique private key set allocation for each user is then shown in Table 2.

In this scenario, we know that

- Each person keeps a predetermined subset of private keys, and no one else has all the private keys in that subset.
- For a public-private key pair, multiple copies of the private key can be held by different users. In the given scenario, each private key has 4 copies.

Table 2. An example private key allocation

User	$\mathcal{K}_i^{priv}$ private-key set held by user i
1	$\mathcal{K}_1^{priv} = \{k_{priv}^1, k_{priv}^2\}$
2	$\mathcal{K}_2^{priv} = \{k_{priv}^1, k_{priv}^3\}$
3	$\mathcal{K}_3^{priv} = \{k_{priv}^1, k_{priv}^4\}$
4	$\mathcal{K}_4^{priv} = \{k_{priv}^1, k_{priv}^5\}$
5	$\mathcal{K}_5^{priv} = \{k_{priv}^2, k_{priv}^3\}$
6	$\mathcal{K}_6^{priv} = \{k_{priv}^2, k_{priv}^4\}$
7	$\mathcal{K}_7^{priv} = \{k_{priv}^2, k_{priv}^5\}$
8	$\mathcal{K}_8^{priv} = \{k_{priv}^3, k_{priv}^4\}$
9	$\mathcal{K}_9^{priv} = \{k_{priv}^3, k_{priv}^5\}$
10	$\mathcal{K}_{10}^{priv} = \{k_{priv}^4, k_{priv}^5\}$

- A message is encrypted by multiple public keys, and it can only be read by a user who has the corresponding private keys. For example, if user 1 encrypts a message m by public keys k_{pub}^2 and k_{pub}^5 as $Enc(Enc(m, k_{pub}^2), k_{pub}^5)$, then only user 7 can decrypt it with private keys k_{priv}^2 and k_{priv}^5.

In traditional public management schemes, each user holds one public-private key pair. Therefore, a user should store n public keys and 1 private key to achieve self-contained key management in a network of size n. In *SMOCK* 10-user example, a user only needs to store 7 keys (5 public keys and 2 private keys), which is smaller than 11 keys (10 public keys and 1 private keys) in traditional schemes. We will show that in *SMOCK* the total number of keys held by each user is approximately $O(log(n))$, but it is $O(n)$ under traditional key management schemes.

3.1 Definitions

Definition 1: Let us consider a *key pool* $\mathcal{K} = \{(k_{pub}^i, k_{priv}^i) | \forall i \leq a\}$, where the i-th public-private key pair is defined as (k_{pub}^i, k_{priv}^i), and $a = |\mathcal{K}|$ represents the number of distinct key pairs. The symbol $\mathcal{K}_v^{priv}$ and $\mathcal{K}_v^{pub}$ stand for a set of private keys and a set of public keys held by user v respectively.

Definition 2: A *key allocation* KA: $2^{\mathcal{K}} \to V$, maps the key pairs in $\mathcal{K}$ to a set of users in V, so that $v \in V$ is assigned a subset of key pairs $\mathcal{K}_i$ ($\mathcal{K}_i \subset \mathcal{K}$)). To guarantee the secure communication between each pair of nodes i and j, we have $\forall i\ \forall j\ \mathcal{K}_i \not\subseteq \mathcal{K}_j$ (the same as $\mathcal{K}_i^{priv} \not\subseteq \mathcal{K}_j^{priv}$) and $\mathcal{K}_j \not\subseteq \mathcal{K}_i$ (the same as $\mathcal{K}_j^{priv} \not\subseteq \mathcal{K}_i^{priv}$), $iff\ i \neq j$. If this property holds, the *key allocation* is valid.

Definition 3: We say that a key allocation is isometric, if $|\mathcal{K}_1| = |\mathcal{K}_2| = \cdots = |\mathcal{K}_n| = b$; otherwise, the key allocation is non-isometric.

Definition 4: We say that the key assignment to user i and j conflict, if either $\mathcal{K}_i^{priv} \subseteq \mathcal{K}_j^{priv}$ or $\mathcal{K}_j^{priv} \subseteq \mathcal{K}_i^{priv}$. For a valid key allocation, there does not exist conflicting key assignments for any pair of the users.

3.2 Objectives

To guarantee the secure communication among n people, we need to have enough public-private key pairs. On the other hand, to seek efficiency in storage and computation, we want to use a small number of key pairs and distribute a small number of key copies to each person. Generally, we desire the key management to be memory efficient for key storage, computationally efficient during encryption and decryption, and resilient to break-ins. Therefore, we define multiple objectives of the *SMOCK* key allocation mechanism as follows:

Objective 1 *Memory Efficiency*: Given a network of size n, we need to find a *key pool* $\mathcal{K}$ and a *key allocation* KA to achieve

$$\begin{cases} min & |\mathcal{K}| + \max_{i \in V} |\mathcal{K}_i^{priv}| \\ s.t. & \mathcal{K}_i \not\subseteq \mathcal{K}_j \text{ and } \mathcal{K}_i \not\supseteq \mathcal{K}_j \quad \forall i \neq j \end{cases} \tag{1}$$

where $|\mathcal{K}_i^{priv}| = |\mathcal{K}_i|$ is the total number of private keys stored at node i. $|\mathcal{K}|$ is the total number of public keys stored at each node. Note that each node stores all public keys before the node deployment, but it only stores a small subset of private keys $\mathcal{K}_i^{priv}$ for user i. If a user is assigned a key pair (k_{pub}, k_{priv}), then the user holds the private key k_{priv}. Therefore, $|\mathcal{K}| + |\mathcal{K}_i^{priv}|$ is the number of memory slots at node i to store the public keys and private keys for secure communications.

Objective 2 *Computational Complexity*: To simplify security operation, each person wants to use a small number of public keys to encrypt the outgoing messages, and a small number of private keys to decrypt incoming messages. Therefore, we have the following objective

$$\begin{cases} min & \max_{i \in V} |\mathcal{K}_i^{priv}| \\ s.t. & \mathcal{K}_i \not\subseteq \mathcal{K}_j, \mathcal{K}_i \not\supseteq \mathcal{K}_j\ (\forall i \neq j) \text{ and } |\mathcal{K}| \leq M \end{cases} \tag{2}$$

where M is the total number of memory slots for key storage at each node.

Proposition 1: Isometric allocation of keys performs better than non-isometric allocation in terms of *Objective 1* and *Objective 2*.

Proof of *Proposition 1* is shown in [21]. Therefore, we assume isometric key allocation throughout the rest of this paper.

Objective 3 *Resilience Requirement*: Under isometric key allocation scheme, we denote $a = |\mathcal{K}|$ and $b = |\mathcal{K}_i| =$

$|\mathcal{K}_i^{priv}|$. Each user needs only to carry b private keys and a public keys under isometric key allocation, wherein $b << a << (a+b) << n$. Clearly, if a node is compromised, all its keys are compromised, regardless of the number of private keys it carries. Therefore, on the average $C(k_c(x), b)$ distinct key-sets are compromised when adversaries break into x nodes, and up to $C(k_v(x), b)$ distinct key-sets are compromised in the worst case.

We denote a vulnerability metric by $V_x(a, b)$, which is the percentage of communications being compromised when x nodes are broken in. It follows that the vulnerability metric, $V_x(a, b)$ is $\frac{C(k_c(x),\,b)}{C(a,b)}$ on the average or $\frac{C(k_v(x),\,b)}{C(a,b)}$ in the worst case, where $k_v(x) = xb$. To achieve the desired resilience when adversaries break into x nodes, we define the resilience requirement as

$$V_x(a,\ b) = \frac{C(k_c(x), b)}{C(a, b)} \leq \mathcal{P} \tag{3}$$

where $\mathcal{P}$ is the resilience bound representing the upper-bound of the compromised communications when x nodes are randomly compromised, each with equal likelihood.

Proposition 2: Let us assume the number of key pairs used by the network is a, and each node possesses b private keys. If x nodes are broken in, then on average $k_c(x) = \lfloor a-(a-b)\left(\frac{(a-b)}{a}\right)^{x-1}\rfloor$ keys will be disclosed. Therefore, $\frac{C(k_c(x),\,b)}{C(a,b)}$ percentage of the node will be compromised.

Proof of *Proposition 2* can be found in [21]. We can also conclude that in worst case, $k_v(x) = min(xb, a)$ keys will be disclosed, when x nodes are broken in.

We observe that $C(k_c(x), b)$ and $C(k_v(x), b)$ do not compare favorably with x. But, by increasing the value of a, we can make $C(a, b) >> n$, therefore, make $V_x(a, b)$ compare favorably with x/n, which we refer to as *benchmark resilience*. There is a trade-off between memory usage and resilience against break-ins: For a larger number of public-private key pairs, we can get better resilience against break-ins at the cost of larger memory footprint.

4 Key Allocation Algorithm

Due to *Proposition 1*, *SMOCK* uses the isometric key allocation algorithms to achieve the objectives outlined in Section 3.2. In this section we show: (1) For a given network, how to determine a and b; (2) How to allocate distinct private key sets to users to achieve secure communication between each pair of users. To determine value of a and b, we first specify an algorithm to obtain the optimal key allocation solution in terms of both *Objective 1* and *Objective 2* with constraint of the resilience requirement specified in *Objective 3*. Observing the trade-off between memory usage and resilience against break-ins, we then present an algorithm to fully utilize memory space to achieve better resilience by slightly relaxing the optimality of *Objective 1* and *Objective 2*. With the given value of a and b, Section 4.2 discusses key allocation details of *SMOCK*.

4.1 Derivation of a and b

4.1.1 Optimization of design objectives

Value b affects the complexity of encryption and decryption. Therefore, we'd like to relax a to allow b to be small. The extreme case is that $a = n$ and $b = 1$,[2] where each person keeps a key and every key only has a single copy. The following algorithm helps to determine a and b to achieve the design objectives. Assume the network size is n.

Objective 1 requires $\frac{a}{n}$ to be small for key storage efficiency. Meanwhile *Objective 3* requires $\frac{a}{b}$ to be large for good resilience. Therefore, there are two conflicting objectives. *Algorithm 1* trades off between memory efficiency and good resilience.

Algorithm 1:

```
(1) Initialize $l = 2$.
    While $(C(l, \lfloor \frac{l}{2} \rfloor) < n)$
        do $\{l = l + 1\}$;
    $a = l, \quad b = \lfloor \frac{l}{2} \rfloor$;

(2) While $(C(a, b-1) > n)$
        do $\{b = b - 1\}$;

(3) While $(C(a+1, b-1) > n)$
        do $\{a = a + 1, \quad b = b - 1\}$

(4) While (Equation (3) is not satisfied)
        do {
            if($C(a+1, b-1) > n$)
                $\{a = a + 1, \ b = b - 1\}$
            else
                $\{a = a + 1\}$
        }
(5) $|\mathcal{K}| = a \ \ and \ \ |\mathcal{K}_i| = b$.
```

Step (1) of Algorithm 1 calculates the minimum number of memory slots to store public keys in order to support the secure communication among n nodes. Step (2) minimizes *Objective 1*. Step (3) further optimizes the *Objective 2* while keeping *Objective 1* unchanged. Step (4) ensures that the key allocation meet *Objective 3*. If the resulting a and b do not satisfy the resilience requirement specified by *Objective 3*, we either increase a, or simultaneously increase a and decrease b. Thus $\frac{a}{n}$ is increased by $\frac{1}{n}$ and $\frac{a}{b}$ is increased by $\frac{1}{b}$ or $\frac{b+1}{b(b-1)}$. For $n >> b$, it is a reasonable trade-off of memory slots to achieve better resilience.

[2]This is exactly the traditional public key management scenario.

4.1.2 Meeting key storage constraint

Total memory slots for key storage are often limited by M, where M is large enough to support n nodes. In this case, we should fully utilize the memory slots to optimize *Objective 2* and achieve the best resilience given by $\frac{C(k_c(x),b)}{C(a,b)}$ in Equation 3. Thus, we come up with Algorithm 2.

```
Algorithm 2:
(1) Let $a = \lceil \frac{2M}{3} \rceil,\ b = \lfloor \frac{M}{3} \rfloor$;
(2) While $(C(a+1, b-1) > n)$
        do $\{a = a+1,\ b = b-1\}$;
(3) Then $|\mathcal{K}| = a\ and\ |\mathcal{K}_i| = b$.
```

4.2 Key Allocation

For a given network size n, we have determined a and b. The key assignment should satisfy $\mathcal{K}_i \not\subseteq \mathcal{K}_j$ and $\mathcal{K}_i \not\supseteq \mathcal{K}_j$, so that the *key allocation* described above can support the pair-wise secure communication for a network of size $n = C(a, b)$. Assuming a single private key can be assigned to at most y nodes, we have $b \times n = a \times y$ (both sides indicate the total copies of private keys in the system). Therefore, $y = \frac{b}{a} n = \frac{b}{a} C(a, b)$. We randomly assign b private keys to network nodes in the key allocation, where a single key should be assigned to at most $\frac{b}{a} C(a, b)$ nodes. Otherwise, we cannot get a valid key allocation. In the example given in Table 2, each key is assigned 4 times, where $a = 5$, $b = 2$. For a key assignment, we just need to assign a random unused private key combination to a node (totally, there are $C(a, b)$ possible combinations). Algorithm 3 illustrates the procedure to assign a subset private keys to a node. Note that very small a and b can support a very large network. E.g., if we ignore the resilience requirement, $a = 20, \quad b = 4$, the network size can be as large as 4845.

```
Algorithm 3:
(1) For the $i$-th node $(i \le C(a, b))$, randomly select $b$
distinct private keys to generate a subset of keys, where
either of these $b$ private keys has been assigned more
than $\frac{b}{a} C(a, b)$ times;
(2) If (the generated key set = an assigned key set)
        Adjust key by key in the generated key set to get
unassigned key set;
(3) Assign the generated key set to node $i$.
```

5 Secure Communication Protocols

Section 4 shows how to determine a, b and how to assign private-key set to a node if network size n is given. In this section we specify detailed protocols used for initialization, communication, and bootstrapping when new nodes are deployed. The initialization phase is performed before deployment. Since communication and bootstrapping are on-line procedures, they have to be very efficient in terms of communication overhead (using a small number of messages).

5.1 Initialization

The initialization phase is to assign keys and identifications to each node. The algorithms for key allocation are shown in Section 4. A node's identification (ID) is a good indicator to show what subset of private keys the node carries. If two nodes want to exchange a secure message, each needs to know the ID of the other. From the ID, a node can infer which private keys the other node has, and it can encrypt the message with the corresponding public keys. Node IDs do not have to form a contiguous range. After key allocation, each node knows the private keys assigned to it, and all the public keys. We label the keys by numbers $0, 1, 2 \cdots$. Let $keyID_i^j$ be the i-th private key held by node j. Let a be the total number of public keys and b be the number of private keys kept at each node. For each node j, we have $keyID_1^j < keyID_2^j < \cdots < keyID_b^j$. The ID field spans $b \times \lceil \log_2 a \rceil$ bits as shown in Figure 1. Each $keyID_i^j$ takes $\lceil \log_2 a \rceil$ bits. It is easy to show that the node ID is unique as long as each node is assigned a unique subset of private keys.

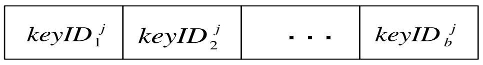


Figure 1. ID field of node j

In the example shown in Table 2, user 7's private key set is $\mathcal{K}_7^{priv} = \{k_{priv}^2, k_{priv}^5\}$. Correspondingly, the ID of the user 7 is "$010|101$", where $a = 5$, $b = 2$. We can see that a node automatically obtains an ID after it has been assigned a private-key set. If other peer nodes know user 7's ID, they can infer that user 7 has private key number 2 (k_{priv}^2) and private key number 5 (k_{priv}^5). If user 7 claims a fake identity, other nodes will use public keys represented by the fake identity to encrypt the messages. Therefore, the user 7 cannot decrypt the message. In this way, *SMOCK* scheme is able to resist against the *Sybil* attack.

5.2 Secure Communication

Figure 2 shows a protocol of secure communication between Alice and Bob, where Alice and Bob establish a secure communication channel. If Alice already knows Bob's ID, she can send an encrypted message (EncMsg) directly to Bob. Otherwise, she needs to send a ID request message to Bob, and Bob replies with his ID. After Alice receives

Bob's ID, she can figure out which private keys Bob is associated with, and she encrypts the message correspondingly before she sends the message.

Since Bob holds a unique subset of private keys, only he is able to decrypt the message correctly. Note that, Bob's ID can be transmitted by plain text. Even so, malicious users who steal Bob's ID cannot decrypt the encrypted message.

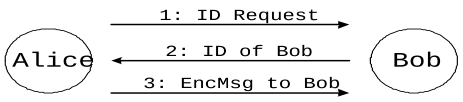


Figure 2. Secure communication protocol between Alice and Bob

5.3 Bootstrapping to Accommodate New Nodes

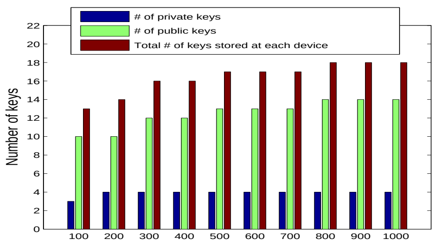
In some cases, we need to deploy new nodes to an existing ad hoc network. In *SMOCK*, it is trivial to make newly deployed node to trust previously deployed nodes. However, in case of insufficient number of keys, a bootstrapping procedure should be run to have previously deployed devices trust newly deployed devices. Let us assume that n nodes are already deployed in a network with a public keys and each node stores b private keys, and m new nodes are being assigned into the network. If $n + m < C(a, b)$ and resilience requirement (Equation (3)) are still satisfied after we deploy m more nodes, then no bootstrapping is necessary, since the newly deployed nodes can be assigned with unused combinations of private keys from the existing key pool owned by off-line trusted server before they are deployed. However, if network size $n + m$ is larger than $C(a, b)$ or resilience requirement is violated after incremental deployment, then the system needs to generate more key pairs, say a' new key pairs. We can still assign b private keys to the additional nodes before their deployment. In this case, a bootstrapping procedure is necessary to introduce newly generated public keys to the previously deployed nodes is necessary. After new nodes join the network, they need to broadcast the newly generated public keys to those previous deployed nodes. Therefore, the previous deployed nodes are able to infer the value of $a+a'$. To prevent unauthorized nodes to broadcast fake public keys in the bootstrapping procedure, the trusted domain center should sign the newly generated public keys. Since we fix b, those previously deployed nodes can adjust the existing ID field to span $b \times \lceil \log_2(a + a') \rceil$ bits.

It can be verified that, given $C(a, b)$, the increment of a by 1 brings $C(a, b - 1)$ new valid key sets for new nodes. Therefore, with a' new key pairs, the network is able to accommodate $\sum_{i=0}^{a'-1} C(a + i, b - 1)$ new nodes. Note that keeping b unchanged and increasing a does not violate the resilience bound $\mathcal{P}$ given in *Objective 3*.

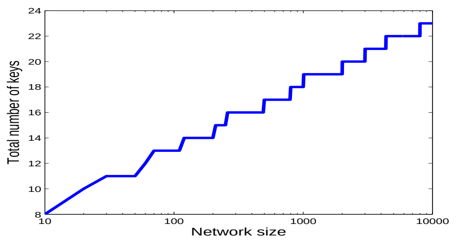
6 Evaluations

6.1 Small Memory Footprint

In *SMOCK*, a few key pairs can support secure communication of a very large network. According to the *Algorithm 1* in Section 4.1, 18 key pairs in the network can support end-to-end secure communication among up to 1000 nodes without resilience consideration. In Figure 3(a), we show the minimum number of keys needed at each node for typical mission-critical network sizes. Therefore, we can achieve very small memory footprint under the *SMOCK* scheme.



(a) $n \le 1000$

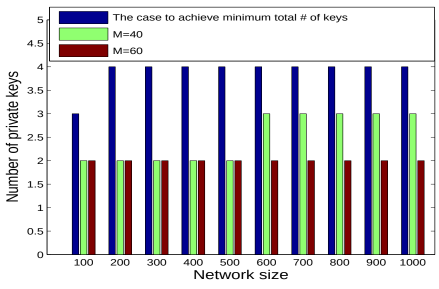


(b) On logarithmic scale $n \le 10000$

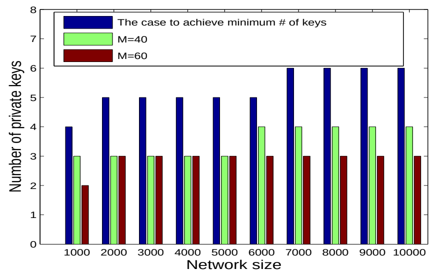
Figure 3. The minimum number of keys needed

A total of a public keys can support at most $C(a, \lfloor \frac{a}{2} \rfloor)$ nodes in the network. By Stirling's Approximation, $n! \approx \sqrt{(2n + \frac{1}{3})\pi}\ n^n e^{-n}$. Hence, a public keys can support a network of size $\Theta(\frac{2^a}{\sqrt{a}})$, where 2^a is dominant as n turns very large. Accordingly, the total number of key pairs required is at a level of $\Theta(\log_2 n) = \Theta(\frac{1}{\lg 2} \lg n) = \Theta(\lg n)$, which can be verified by Figure 3(b). We conclude that the *SMOCK* scheme yields very small memory footprint.

If we relax the storage limitation, the number of private keys needed decreases, and computational complexity is reduced accordingly. Figure 4 shows the trade-off between computational complexity and key storage space for different network scales, where the computational complexity is inferred by the number of private keys needed. We can conclude that the larger the storage space is, the smaller number of private keys are kept at each node, thus the smaller computational complexity it is.



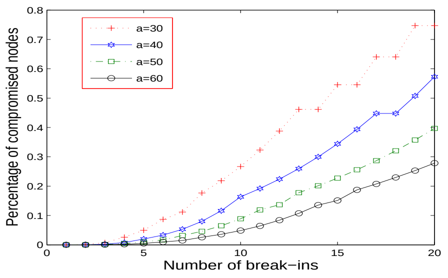
(a) $n \leq 1000$



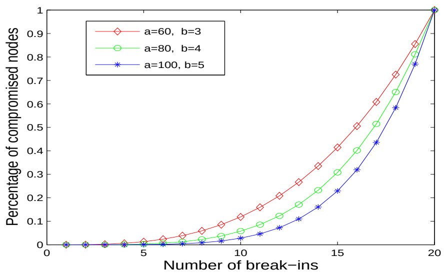
(b) $1000 \leq n \leq 10000$

Figure 4. Trade-off between storage space and computational complexity (M is total memory slots for key storage)

of multiple nodes may compromise a set of other nodes. Assume x nodes are compromised and $k_c(x)$ is the expected number of keys disclosed correspondingly. As *Proposition 2* shows, $k_c(x) = \lfloor a - (a-b)\left(\frac{a-b}{a}\right)^{x-1} \rfloor$. Then $\frac{C(k_c(x),\, b)}{C(a,b)}$ percentage of nodes will be compromised. Let's assume $n = 1000$, Figure 5(a) shows the average case percentage of compromised nodes when a small portion of nodes are controlled by adversaries.



(a) Average case ($b = 4$)



(b) Worst case

Figure 5. Percentage of compromised nodes with break-ins

6.2 Communication Overhead for Key Management

Since *SMOCK* is a self-contained public-key management scheme, a node does not need to contact/trust other nodes for certificate verification. Only during the bootstrapping phase when new nodes join the network and the key revocation process, communication is needed for key management. Therefore, *SMOCK* has little communication overhead for key management.

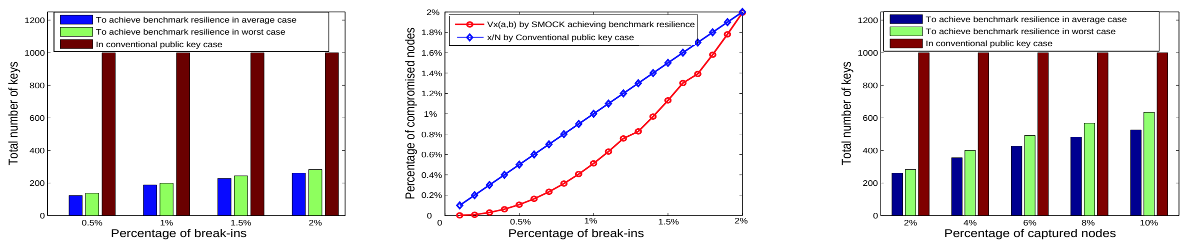
6.3 Resilience to Break-ins

6.3.1 Average case analysis

The break-in of any single node by an adversary does not release enough information to the adversary to break secure communication for any pair of nodes. However, break-ins

6.3.2 Worst case analysis

For mission-critical applications, it may be important to consider resilience against the worst case where each newly compromised node releases b new keys to the adversary. If we define $k_v(x)$ as the number of keys disclosed by the break-ins of x nodes, then in the worst case, $k_v(x) = min(xb, a)$, where a is the total number of key pairs and b is the number of private keys kept by each node. In the worst case, we want to calculate the probability that an allocated key set is compromised as $Prob(a\, key\, set\, is\, compromised \,|\, the\, key\, set\, is\, allocated)$. Since the events "a key set is compromised" and "a key set is allocated" are independent, then the worst case probability is $Prob(a\ key\ set\ is\ compromised)$. Therefore, given a and b, in worst case, the break-in of x nodes results in $\frac{C(k_v(x),\, b)}{C(a,b)}$ percent of the communication compromises, where n is the network size. Figure 5(b) shows the worst



(a) Smaller storage space ($x \leq 20$)

(b) Equivalent resilience when achieving benchmark resilience at $x = 20$

(c) Storage space to achieve benchmark resilience ($20 \leq x \leq 100$)

Figure 6. Comparison between *SMOCK* which achieves benchmark resilience and conventional public key scheme for $n = 1000$

case percentage of the compromised nodes, where we can see that the break-in of $\lceil \frac{a}{b} \rceil$ nodes can compromise the whole network in the worst case. However, the break-in of $\lceil \frac{a}{2b} \rceil$ nodes only compromises a small ratio of the network. With the help of break-in detection and key revocation mechanisms[3] we can assume that only a few number of nodes (less than $\lceil \frac{a}{2b} \rceil$) can be broken in.

6.3.3 Control resilience

As long as the number of key pairs is large enough, the percentage of the compromised nodes will be small enough when a certain number of nodes are compromised. This is the practical reason that we want to choose a somewhat larger value for a, the total number of key pairs used in the network. Figure 5 shows that break-in of any single node cannot compromise any other node in the network, and break-in of multiple nodes may disclose information to the adversary to compromise more than the number of nodes which are broken in. The break-in of multiple nodes will be more expensive for the adversary than the break-in of a single node. On the other hand, whenever the network detects the compromise of a user, it is necessary to nip it in the bud by dynamically revoking and redistributing new keys. Consider the resilience requirement as: $V_x(a, b) = \frac{C(k_c(20),\, b)}{C(a,b)} \leq 20\%$. When 20 or fewer nodes are cracked in, we require that at most 20% of the secure channels are compromised. According to *Algorithm 1*, the minimum number of memory slots needed to fulfill such resilience requirement is 70.

Figure 6 compares the total number of keys needed to achieve $V_x(a, b) \approx x/n$ for $x = 20$ under *SMOCK* with the conventional public key scheme. We assume that only a small subset of nodes may be broken in during a reasonable time window before key revocation. Figure 6(a) and Figure 6(b) show that in the case *SMOCK* achieves benchmark resilience at $x = 20$ that $V_x(a, b) \approx x/n$, it provides equivalent resilience when less than 20 nodes are compromised, but requires much smaller memory size, comparing with conventional scheme. Figure 6(c) shows the total number of keys required to be stored at each node in order to achieve benchmark resilience when x goes up until $x = 100$. It shows good scalability of *SMOCK* to tolerate more break-ins. For applications with a high resilience requirement, we recommend using x/n as the resilience bound in *Objective 3*.

[3]Note that break-in detection and key revocation mechanisms are important mechanisms, which are out the scope of this paper.

7 Conclusions

We depict a key self-contained key management scheme, which requires significantly less key storage space than traditional schemes and almost zero communication overhead for authentication in a mission-critical wireless ad hoc network with n nodes. The scheme also achieves controllable resilience against node compromise by defining required benchmark resilience.

In this paper, we generalize the traditional public key management schemes. $a = n$ and $b = 1$ in *SMOCK* turned out to be the traditional public-key infrastructure. We can also see that *SMOCK* scheme is able resist *Sybil* attacks, and fulfill the secure communication requirement in terms of *integrity*, *authentication*, *confidentiality*, *non-repudiation*, and *service availability*.

8 Acknowledgement

The research in this paper is supported by Motorola grant 1-557641-239016-191100.

References

[1] A. Menezes, P.V. Oorschot, and S. Vanstone, *Handbook of Applied Cryptography.* CRC Press, Boca Raton, FL, 1996.

[2] B. Schneier, *Applied Cryptography.* 2nd Edition, John Wiley & Sons, New York, 1996.

[3] L. Zhou and Z. J. Haas. *Securing Ad Hoc Networks.* IEEE Network Magazine, Nov. 1999.

[4] J. Douceur. *The Sybil Attack.* In Proceedings of the IPTPS Workshop, March 2002.

[5] S. Kent and T. Polk. *Public-key infrastructure (x.509) (pkix) charter.* Available at http://www.ietf.org/html.charters/pkixcharter.html.

[6] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. *Providing robust and ubiquitous security support for mobile ad-hoc networks.* In Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP 2001), 2001.

[7] S. Capkun, L. Buttyan, and J.-P. Hubaux. *Self-organized public-key management for mobile ad hoc networks.* IEEE Transactions on Mobile Computing, 2(1), January-March 2003.

[8] A. Cheng and E. Friedman. *Sybilproof Reputation Mechanisms*. In ACM Workshop on the Economics of Peer-to-Peer Systems, August 2005.

[9] M. G. Gouda, and E. Jung, *Certificate Dispersal in Ad-Hoc Networks*, in the Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS 04), March 2004.

[10] S. Yi, R. Kravets, *MOCA: Mobile Certificate Authority for Wireless Ad Hoc Networks*, 2nd Annual PKI Research Workshop (PKI '03), Gaithersburg, Maryland, April, 2003.

[11] S.A. Camtepe and B. Yener, *Combinatorial design of key distribution mechanisms for wireless sensor networks.* In Proceedings of 9th European Symposium On Research in Computer Security (ESORICS 04), 2004.

[12] L. Eschenauer and V. D. Gligor. *A key-management scheme for distributed sensor networks.* In Proceedings of the 9th ACM Conference on Computer and Communications Security, pages 41-47, November 2002.

[13] H. Chan, A. Perrig, and D. Song. *Random key predistribution schemes for sensor networks.* In IEEE Symposium on Research in Security and Privacy, pages 197-213, 2003.

[14] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. *A pairwise key pre-distribution scheme for wireless sensor networks.* In Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), pages 42-51, Washington, DC, USA, October 27-31 2003.

[15] S. Zhu, S. Setia, and S. Jajodia. *LEAP: Efficient security mechanisms for large-scale distributed sensor networks.* In Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03), pages 62-72, October 2003.

[16] F. Delgosha and F. Fekri. *Threshold Key-Establishment in Distributed Sensor Networks Using a Multivariate Scheme.*In Proceedings of 25th IEEE INFOCOM, April 2006.

[17] P. Traynor, H. Choi, G. Cao, S. Zhu, T. La Porta. *Establishing Pair-Wise Keys in Heterogeneous Sensor Networks.* In Proceedings of 25th IEEE INFOCOM, April 2006.

[18] G. Gaubatz, J. Kaps, and B. Sunar. *Public keys cryptography in sensor networks — revisited.* In The Proceedings of the 1st European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS), 2004.

[19] D. J. Malan, M. Welsh, and M. D. Smith. *A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography.* In The First IEEE International Conference on Sensor and Ad Hoc Communications and Networks, Santa Clara, California, October 2004.

[20] V. Gupta, M Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. Chang. *Sizzle: A Standards-based End-to-End Security Architecture for the Embedded Internet.* In In Proceedings of the 3rd IEEE Percom 2005.

[21] W. He, Y. Huang, K. Nahrstedt, W. C. Lee, *SMOCK: A Scalable Method of Cryptographic Key Management For Mission-Critical Networks.* UIUC Techniqual Report in Department of Computer Science, UIUCDCS-R-2006-2734.