SMOCK: A Scalable Method of Cryptographic Key Management for Mission-critical Wireless Ad Hoc Networks

Wenbo He †, Ying Huang ‡, Ravishankar Sathyam †, Klara Nahrstedt †, Whay C. Lee ‡

1) Dept. of Computer Science
University of New Mexico
Albuquerque, NM 87131

† Dept. of Computer Science
University of Illinois at Urbana-Champaign
Champaign, IL, 61801

‡ Motorola Labs
111 Locke Drive
Marlborough, MA 01752

Abstract—Mission-critical networks show great potential in emergency response and/or recovery, health care, critical infrastructure monitoring, etc. Such mission-critical applications demand security service be “anywhere”, “anytime” and “anyhow”. However, it is challenging to design a key management scheme in current mission-critical networks to fulfill the required attributes of secure communications, such as data integrity, authentication, confidentiality, non-repudiation and service availability. In this paper, we present a self-contained public key management scheme, called SMOCK, which achieves almost 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 compromise a limited number of nodes before key revocation and renewal.

I. INTRODUCTION

With the advances in cost-effective sensing, computing, and communication wireless devices, current mission-critical systems are composed of mobile, autonomous, wireless devices. Examples can be found in health care (assisted living) systems, automotive networks, first responder systems (emergency rescue and disaster recovery), military applications, critical infrastructure monitoring, and so on. In these systems, it is important to support secure communications, such as data integrity, authentication, confidentiality, non-repudiation and service availability. To build a secure communication system, usually the first attempt is to employ cryptographic keys. However, the cryptographic key management is challenging due to the following characteristics of wireless ad hoc communications:

1) Unreliable Communications and Limited Bandwidth: 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. Therefore, the communication overhead for certificate exchange cannot be ignored.

2) Network Dynamics: 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 resources 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), network nodes may encounter untrustworthy peers from time to time due to node mobility and unreliable communication. In certificate-based schemes, certificate exchange may incur noticeable message overhead in resource (bandwidth) limited environments. Therefore, we need a self-contained key management scheme, which allows a mobile node to contain all the necessary information for authentication locally. 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. Figure 1 shows an example scenario of a mission-critical network.

The common criticism on using PKC is its computational complexity. However, recent studies [22] [23] [24] show how practical PKC can be used for mobile and resource-limited networks. In this paper, we design a self-contained public-key management scheme, where all necessary cryptographic keys 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, because it does not require the exchange of certificates in communication. Rather, nodes need to know the ID of the other party in
communication to infer public keys of each other. In contrast to traditional certificate-based schemes, the authentication procedure of SMOCK does not require certificate exchange. 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 II, we summarize the related work in key management schemes in mobile ad hoc networks. In Section III, we describe the background and problem description. Section IV provides the details of our key allocation algorithms. Section V gives detailed protocols for secure communication and bootstrapping when new nodes are deployed. Section VI evaluates the proposed scheme. Finally, Section VII provides concluding remarks.

II. Related Work

For secure communication, wireless sensor networks use symmetric key techniques [14], [15], [16], [17], [18], [19], [20], [21]. 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, [15] and [16] are not suitable in mission-critical applications over wireless ad hoc networks. Pairwise key distribution schemes [17] [18] and [20] 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 does not provide the verifiability property [9], hence 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, Buttyn, 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] [12] [13]. Zhu et al, present two efficient and robust key management schemes for large scale mobile ad hoc networks [10] to resist active attacks. The certificate based schemes presented in [10] are designed for dynamic networks and able to adapt to changing topology of networks with efficient memory usage. But it requires the help from neighboring nodes for authentication. In this case, if the number of illegitimate nodes is larger than a threshold, they may generate valid certificates through the collusion. Therefore, we turn to self-contained public key approaches for high security assurance and low communication overhead. In our self-contained key management approach, illegitimate nodes need to compromise enough legitimate nodes before they impersonate non-exist nodes.

Identity-based cryptography [11] facilitates public key cryptography by allowing us derive an entity’s public key from its identity. It reduces the need for public key certificates. However, identity-based cryptography does not suit the case if timely revocation is a strong requirement, because the fine-grained key updates may introduce large communication overhead.

Recent development of ECC algorithms and implementations [22] [23] [24] 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.

III. 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. 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 $(k_{\text{priv}}^i, k_{\text{pub}}^i)$. 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 $\mathcal{K}_{\text{priv}}^{\text{Alice}}$ denote a subset of private keys held by Alice, and $\mathcal{K}_{\text{pub}}^{\text{Alice}}$ represents Alice’s corresponding public key subset. If Bob wants to send a secret message to Alice, he needs to know $\mathcal{K}_{\text{pub}}^{\text{Alice}}$, where $\mathcal{K}_{\text{priv}}^{\text{Alice}} \in \mathcal{K}_{\text{anybody,else}}^{\text{priv}}$. Bob is able to pass the secret message to Alice, using the public keys $\mathcal{K}_{\text{pub}}^{\text{Alice}}$ to encrypt the message. The message can be opened only by Alice, who has the private key set $\mathcal{K}_{\text{priv}}^{\text{Alice}}$, but others do not.

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 $(k_{\text{priv}}^1, k_{\text{pub}}^1), (k_{\text{priv}}^2, k_{\text{pub}}^2), (k_{\text{priv}}^3, k_{\text{pub}}^3), (k_{\text{priv}}^4, k_{\text{pub}}^4), (k_{\text{priv}}^5, k_{\text{pub}}^5)$. Each user keeps 5 public keys and 2 private keys. The unique private key set allocation for each user is then shown in Table I.

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.
- 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 $k_{\text{pub}}^2$ and $k_{\text{pub}}^5$ as $\text{Enc}(\text{Enc}(m, k_{\text{pub}}^2), k_{\text{pub}}^5)$, then only user 7 can decrypt it with private keys $k_{\text{priv}}^2$ and $k_{\text{priv}}^5$.

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

---

**TABLE I**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{K}$</td>
<td>A Key pool: a set of public-private key pairs</td>
</tr>
<tr>
<td>$\text{privateKey}_{ij}$</td>
<td>$j$-th private key held by user $i$</td>
</tr>
<tr>
<td>$\text{publicKey}_{ij}$</td>
<td>$j$-th public key held by user $i$</td>
</tr>
<tr>
<td>$\mathcal{K}_{\text{priv}}^i$</td>
<td>A set of private keys held by user $i$, $\mathcal{K}<em>{\text{priv}}^i = {\text{privateKey}</em>{ij}}$</td>
</tr>
<tr>
<td>$\mathcal{K}_{\text{pub}}^i$</td>
<td>A set of public keys corresponding to $\mathcal{K}_{\text{priv}}^i$</td>
</tr>
<tr>
<td>$\mathcal{K}_i$</td>
<td>A set of public-private key pairs held by user $i$, $\mathcal{K}<em>i = {(k</em>{\text{priv}}, k_{\text{pub}})</td>
</tr>
<tr>
<td>$M$</td>
<td>Memory size for key storage</td>
</tr>
<tr>
<td>$a$</td>
<td>Number of distinct key pairs $a =</td>
</tr>
<tr>
<td>$b$</td>
<td>Number of private keys held by each user under isometric key allocation, $b =</td>
</tr>
<tr>
<td>$k_c(x)$</td>
<td>Expected number of disclosed keys when $x$ nodes are broken in</td>
</tr>
<tr>
<td>$k_v(x)$</td>
<td>Maximum number of disclosed keys when $x$ nodes are broken in</td>
</tr>
<tr>
<td>$V_x(a, b)$</td>
<td>Vulnerability metrics as $x$ nodes are broken in</td>
</tr>
<tr>
<td>$C(a, b)$</td>
<td>Abbreviation of $a$ choose $b$, $\binom{a}{b}$</td>
</tr>
<tr>
<td>$V$</td>
<td>A set of nodes in the ad hoc wireless network</td>
</tr>
<tr>
<td>$n$</td>
<td>Total number of nodes in the network, $n =</td>
</tr>
</tbody>
</table>

---

2Symbols and terms used throughout this paper are shown as in Table I.
we need to find a multiple objectives of the SMOCK key storage, computationally efficient during encryption and we desire the key management to be memory efficient for we want to use a small number of key pairs and distribute other hand, to seek efficiency in storage and computation, we manage schemes.

TABLE II
AN EXAMPLE PRIVATE KEY ALLOCATION

<table>
<thead>
<tr>
<th>User</th>
<th>(K_i^{ priv} ) private-key set held by user (i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(K_1^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>2</td>
<td>(K_2^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>3</td>
<td>(K_3^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>4</td>
<td>(K_4^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>5</td>
<td>(K_5^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>6</td>
<td>(K_6^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>7</td>
<td>(K_7^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>8</td>
<td>(K_8^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>9</td>
<td>(K_9^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
<tr>
<td>10</td>
<td>(K_{10}^{ priv} = {k_1^{ priv}, k_2^{ priv} } )</td>
</tr>
</tbody>
</table>

approximately \(O(\log(n))\), but it is \(O(n)\) under traditional key management schemes.

A. Definitions

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

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

Definition 3: We say that a key allocation is isometric, if \(|K_1| = |K_2| = \cdots = |K_n| = b\); otherwise, the key allocation is non-isometric.

Definition 4: We say that the key assignment to user \(i\) and \(j\) conflicts, if either \(K_i^{ priv} \subseteq K_j^{ priv}\) or \(K_j^{ priv} \subseteq K_i^{ priv}\). For a valid key allocation, there does not exist conflicting key assignments for any pair of the users.

B. 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 \(K\) and a key allocation \(KA\) to achieve

\[
\begin{align*}
\min\left( |K| + \max_{i \in V} |K_i^{ priv}| \right) \\
\text{s.t. } K_i \not\subseteq K_j \text{ and } K_i \not\supseteq K_j \forall i \neq j
\end{align*}
\]

where \(|K_i^{ priv}| = |K_i|\) is the total number of private keys stored at node \(i\), \(|K|\) 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 \(K_i^{ priv}\) 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, \(|K| + |K_i^{ priv}|\) 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

\[
\begin{align*}
\min\left( \max_{i \in V} |k_i^{ priv}| \right) \\
\text{s.t. } K_i \not\subseteq K_j, K_i \not\supseteq K_j \forall i \neq j \text{ and } |K| \leq M
\end{align*}
\]

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 the Appendix. Therefore, we assume isometric key allocation throughout the rest of this paper.

Objective 3 Resilience Requirement: Under isometric key allocation scheme, we denote \(a = |K|\) and \(b = |K_i| = |K_i^{ priv}|\). Each user needs only to carry \(b\) private keys and \(a\) public keys under isometric key allocation, wherein \(b << a \ll (a + b) \ll n\). Clearly, if a node is compromised, all its keys are compromised, regardless of the number of private keys it carries. Therefore, on the average \(C(k_v(x), b)\) 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 \(V_2(a, b)\), which is the percentage of communications being compromised when \(x\) nodes are broken in. It follows that the vulnerability metric, \(V_2(a, b) = \frac{C(k_v(x), b)}{C(a, b)}\) on the average or \(\frac{C(k_v(x), b)}{C(a, b)}\) in the worst case, where \(k_v(x)\) and \(k_v(x)\) are deduced in Proposition 2. To achieve the desired resilience when adversaries break into \(x\) nodes, we define the resilience requirement as

\[
V_2(a, b) = \frac{C(k_v(x), b)}{C(a, b)} \leq \mathcal{P}
\]

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. Note that we take the floor of \(k_v(x)\), in case \(k_v(x)\) is not an integer.

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 \(k_v(x) = a - (a - b) \left(\frac{a-b}{a}\right)^{x-1}\) keys will be disclosed. Therefore, \(\frac{C(k_v(x), b)}{C(a, b)}\) percentage of the node will be compromised.
Proof of Proposition 2 can be found in the Appendix. We can also conclude that in worst case, \( k_i(x) = \min(xb, a) \) keys will be disclosed, when \( x \) nodes are broken in.

We observe that \( C(k_i(x), a) \) and \( C(k_i(x), b) \) do not compare favorably with \( x \). But, by increasing the value of \( a \), we can make \( C(a, b) >> n \), therefore, make \( V_z(a, b) \) 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.

IV. KEY ALLOCATION ALGORITHM

Due to Proposition 1, SMOCK uses the isometric key allocation algorithms to achieve the objectives outlined in Section III-B. 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 IV-B discusses key allocation details of SMOCK.

A. Derivation of \( a \) and \( b \)

1) Optimization of design objectives: The value of parameter \( b \) 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 \( b = 1 \), 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 \( a \) to be small for key storage efficiency. Meanwhile Objective 3 requires \( a \) to be large for good resilience. Therefore, there are two conflicting objectives. Algorithm 1 trades off between memory efficiency and good resilience.

Algorithm 1:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialize ( l = 2 ), ( a = 0 ), ( b = 0 ). While (( C(l, \lfloor l/2 \rfloor) &lt; n ) do ( l = l + 1 ); ( a = l ), ( b = \lfloor l/2 \rfloor );</td>
</tr>
<tr>
<td>2</td>
<td>While (( C(a + 1, b) &gt; n ) do ( b = b - 1 );</td>
</tr>
<tr>
<td>3</td>
<td>While (( C(a + 1, b - 1) &gt; n ) do ( a = a + 1 ), ( b = b - 1 );</td>
</tr>
<tr>
<td>4</td>
<td>While (Equation (3) is not satisfied) do {</td>
</tr>
<tr>
<td></td>
<td>if (( C(a + 1, b - 1) &gt; n ) then ( {a = a + 1 ), ( b = b - 1 }) else ( {a = a + 1 };</td>
</tr>
<tr>
<td>5</td>
<td>(</td>
</tr>
</tbody>
</table>

Note: \( C(x, y) \) is defined as \( n \) times the number of pairs \((x, y)\) where \( x + y > n \). \( a \), \( b \), and \( n \) are positive integer.

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 meets Objective 3.

In summary, the design goal of Algorithm 1 is to minimize storage space for cryptographic keys while make SMOCK satisfy the given resilience requirement by Equation (3). Algorithm 2 is to use up the specified storage space to achieve the best possible resilience.

B. Key Allocation

For a given network size \( n \), we have determined \( a \) and \( b \). The key assignment should satisfy \( K_i \not\subseteq K_j \) and \( K_i \not\supseteq K_j \), so that the key allocation described above can support the pairwise 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, \( y = \frac{a}{b} n = \frac{2}{a} C(a, b) \). We randomly assign \( b \) private keys to network nodes in the key allocation, where a single key should be assigned to at most \( \frac{a}{b} C(a, b) \) nodes. Otherwise, we cannot get a valid key allocation. In the example given in Table II, 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, \( a = 20 \) and \( b = 4 \), the network size can be as large as 4845.
V. Secure Communication Protocols

Section IV 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 online procedures, they have to be very efficient in terms of communication overhead (using a small number of messages).

A. Initialization

The initialization phase is to assign keys and identifications to each node. The algorithms for key allocation are shown in Section IV. 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 \( \text{keyID}_j^1 \) 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 \( \text{keyID}_{j}^{1} < \text{keyID}_{j}^{2} < \cdots < \text{keyID}_{j}^{b} \). The ID field spans \( b \times \lceil \log_2 a \rceil \) bits as shown in Figure 2. Each \( \text{keyID}^j_i \) 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.

\[
\begin{array}{ccc}
\text{keyID}^1_j & \text{keyID}^2_j & \cdots & \text{keyID}^b_j
\end{array}
\]

Fig. 2. ID field of node \( j \)

In the example shown in Table II, user 7’s private key set is \( K^\text{priv}_7 = \{ k^\text{priv}_7, k^\text{priv}_8 \} \). Correspondingly, the ID of the user 7 is “0101011”, 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 (\( k^\text{priv}_2 \)) and private key number 5 (\( k^\text{priv}_5 \)). 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.

B. Secure Communication

Figure 3 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 an 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.

C. 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 \( a'-1 \sum_{i=0}^{a'-1} C(a+i, b-1) \) new nodes. Note that keeping \( b \) unchanged and increasing \( a \) does not violate the resilience bound \( \mathcal{P} \) given in Objective 3.

D. Key Revocation

In SMOCK, since cryptographic keys are generated and maintained by the central off-line trusted servers, we leave the power to revoke keys as well as create new ones in the hands of central servers. A key certificate revocation message must be spread to all those who might potentially hold it, and as rapidly as possible. Therefore, key revocation in SMOCK relies on message broadcasting, where the revocation messages are signed and pushed by the central servers.

VI. Evaluations

A. 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 IV-A, 18 key pairs in the network can support end-to-end secure communication among up to 1000 nodes without resilience consideration. In Figure 4(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.

Fig. 4. The minimum number of keys needed

A total of \( a \) public keys can support at most \( C(a, \lceil \frac{n}{2} \rceil) \) 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 \( \Theta(\frac{2^a}{\sqrt{n}}) \), where \( 2^a \) is dominant as \( n \) grows very large. Accordingly, the total number of key pairs required is at a level of \( \Theta(\log_2 n) = \Theta(\frac{\log n}{\log 2}) = \Theta(\log n) \), which can be verified by Figure 4(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 5 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.

B. 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.

C. Resilience to Break-ins

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 of multiple nodes may compromise a set of other nodes. Assume \( x \) nodes are compromised and \( k_v(x) \) is the expected number of keys disclosed correspondingly. As Proposition 2 shows, \( k_v(x) = a - (a - b) (\frac{a - b}{a})^x \). Then \( \frac{C(k_v(x), b)}{C(a, b)} \) percentage of nodes will be compromised. Let’s assume \( n = 1000 \), Figure 6(a) shows the average case percentage of compromised nodes when a small portion of nodes are controlled by adversaries.

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 \( \Prob(a \text{ key set is compromised} \mid \text{the key set is allocated}) \). Since the events “a key set is compromised” and “a key set is allocated” are independent, then the worst case probability is \( \Prob(a \text{ key set is compromised}) \). 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 6(b) shows the worst case percentage of the compromised nodes, where we can see that the break-in of \( \lceil \frac{n}{2} \rceil \) nodes can compromise the whole network in the worst case. However, the break-in of \( \lceil \frac{n}{10} \rceil \) nodes only compromises a small ratio of the network. With the help of break-in detection and key revocation mechanisms\(^4\) we can assume that only a few number of nodes (less than \( \lceil \frac{n}{100} \rceil \)) can be broken in.

\(^4\)Note that break-in detection and key revocation mechanisms are important mechanisms, which are out the scope of this paper.
Algorithm 1

20% that at most 20% $V$ achieve the resilience requirement as:

\[ V = \text{dynamically revoking and redistributing new keys.} \]

Consider the compromise of a user, it is necessary to nip it in the bud by node. On the other hand, whenever the network detects the more expensive for the adversary than the break-in of a single node. The break-in of multiple nodes will 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:

\[ V = \frac{C(a,b)}{\binom{a+b}{b}} \leq 20\%. \]

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 7 compares the total number of keys needed to achieve $V(a,b) \approx x/n$ 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 7(a) and Figure 7(b) show that in the case $SMOCK$ achieves benchmark resilience at $x = 20$ that $V(a,b) \approx x/n$, it provides equivalent resilience when less than 20 nodes are compromised, but requires much smaller memory size, comparing with conventional scheme. Figure 7(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.

D. Implementation

We implemented $SMOCK$ under the context of Trustworthy Cyber Infrastructure for the Power grid (TCIP), with C language in Linux operation system. In our implementation, nodes receive their subset of private keys, unique $SMOCK$ IDs, and all $SMOCK$ public keys via SSL channel from trusted authority before secure communication. When a node wants to send a message to another node (the receiver), it sends a plain-text message (along with its $SMOCK$ ID). The receiver then encrypts its $SMOCK$ ID with sender’s public keys, and sends the encrypted message to the sender. The sender can then encrypt the message using the receiver’s $SMOCK$ keys. And the receiver can then decrypt the message using its $SMOCK$ private keys. We measured encryption and decryption process, time taken to encrypt and decrypt a message is show in Table III.

We can easily observe that message length affects encryption and decryption time very little. It takes multiple times as traditional self-certificate schemes to encrypt and decrypt message for $SMOCK$. However, the number of private keys held by nodes in $SMOCK$ is usually small, so the computational overhead of $SMOCK$ is acceptable.

VII. CONCLUSIONS AND FUTURE WORK

We depict a self-contained key management scheme, which adopts combinatorial design of cryptographic keys to achieve lightweight key management. We can further extend the idea of $SMOCK$ on other applications, such as broadcast authentication. Observing that previous work on broadcast authentication using one-way hash chain, represented by TESLA [26] can not scale to a large number of senders. In addition, authentication delay is increased under packet losses and probabilistic broadcast since authentication in TESLA relies on continuous packet arrival from the source to delivery the authentication key. Based on the $SMOCK$ idea, we can design a combinatorial hash-chain sharing scheme: A hash chain pool $HC$ is constructed for the whole network and nodes store the commitment information for all the hash chains in $HC$. Each hash chain is shared by several sources and each source $S$ with $ID$ has a unique set $HCID$, which indicates the hash chains owns by the source. All the hash chains have the same releasing schedule, which is guaranteed by loosely time synchronization. Before network starts, each node pre-stores its assigned one-way hash chains and all the chain commitment information for other sources in memory before being deployed in fields. Message signing and verification use all the hash chains associated with senders’ identity. With the combinatorial design, we expect better scalability and less
delay than traditional broadcast authentication schemes. We will investigate this deeply in future work.

VIII. ACKNOWLEDGMENT

The research in this paper is supported by Motorola grant 1-557641-239016-191100. The authors would like to thank Art Harvey of Motorola for his suggestion to take into consideration difference in key length while evaluating the proposed method.

REFERENCES

[23] D. J. Malan, M. Welsh, and M. D. Smith. A public-key infrastructure for key distribution in TinyOS based on elliptic curve
Appendix

Lemma 1: In a valid non-isometric key allocation $KA$, if node $i$ is assigned the smallest key pair set, and node $k$ is assigned the largest key pair set, then there exists a valid key allocation to increase $b_i$ by 1 with $b_k$ unchanged.

Proof For any other node $j$, we have $b_j \geq b_i$, since node $i$ is assigned the smallest key set. If for all $j$ ($i \neq j$ and $j \in V$), $b_j > b_i$, then we can get a valid key allocation $KA'$ by adding any key pair to user $i$. Since in the new key allocation $KA'$, $K_i \subseteq K_j$ must hold, if it holds in $KA$. In $KA'$, $K_i \subseteq K_j$, which implies $K_i \subset K_j$ in $KA$, thus it contradicts to the fact that $K_i$ is a valid key allocation. Therefore, in this case, we can get a new key allocation $KA'$, which increases $b_i$ by 1, and keeps $b_j$ unchanged.

If there exists a set $J$, where $j \neq i$, and $b_j = b_i$, then we will show how to increase $b_i$ by 1, and still get a valid key allocation $KA'$. If there exists a key pair, adding which to $K_i$ does not cause key assignment conflicting with $i$ and any $j \in J$. Then we can get valid key allocation $KA'$ and increase $b_i$ by 1. If we cannot find such a key pair, then we have $K_i'$ by adding any key pair to $K_i$, and can always find $j \in J$ and $K_j \subseteq K_i'$. In this case, we can replace the original $K_i$ with $K_i'$, and keep the original $K_i$. Such replacement is equivalent to adding a new key pair to $K_j$'s. We can create $(a - b_i - 1)$ potential assignment options by adding one key pair to each $K_j$, $j \in J$. Among all these options, we have totally $\sum k = 1, b_i - 1 k$ distinct new key assignment options, since any two users create a common key assignment option. We have at most $a - b_i$ replacements of $K_j$. Since $\sum k = 1, b_i - 1 k > a - b_i$, so after $a - b_i$ replacement, we still can find an option to replace the original $K_i$. Thus, $b_i$ and $(a - b_i)$ of $b_j$, $j \in J$ increase by 1. So we find a way to increase $b_i$ by one, but keep $b_k$ unchanged.

Lemma 2: Given $K$ and a valid non-isometric key allocation $KA$, we can always find a new valid isometric key allocation $KA'$, so that we can further optimize both Objective 1 and Objective 2 or make the values of objective functions remain the same.

Proof First, we sort the nodes (users) in the increasing order of the number of keys owned by users, so that node 1 keeps the least keys and node $n$ keeps the most keys. If node $i$ keeps a set of keys $K_i = \{k_{i1}, k_{i2}, \ldots, k_{ib_i}\}$, then the number of keys kept by node $i$ is denoted as $|K_i| = b_i$. We have $b_1 \leq b_2 \leq \cdots \leq b_n$. For a valid non-isometric key allocation, there must exist $t$ that $b_{t-1} < b_t$. Choose the smallest $t$. By Lemma 1, there exists a valid key allocation to increase $b_{t-1}$ by 1. The increment on $b_{t-1}$ opens up several possible key assignment options for user $l$, where $b_l > b_l$. If there does not exist such $l$, then the increment on $b_{t-1}$ by 1 doesn’t increase the value of Objective 1 and Objective 2. If there exists such $l$, we know $b_l > b_l$, then we have chance to further optimize Objective 1 and Objective 2 by making $b_l = b_l$.

We can keep doing the above step until we get the isometric key allocation $KA'$. We know that $KA'$ is valid, and Objective 1 and Objective 2 either get smaller or remain the same when comparing to the original non-isometric key assignment $KA$.

Proposition 1: Isometric allocation of keys performs better than non-isometric allocation.

Proof Proposition 1 is intuitive, based on Lemma 1 and Lemma 2.

Given any non-isometric key allocation, there exists an isometric key allocation which achieves the same or smaller value in terms of Objective 1 and Objective 2. On the other hand, given $a$, by isometric key allocation, there doesn’t exist a valid non-isometric key allocation yielding better or equal value in terms of Objective 1 and Objective 2.

Proposition 2: Assume the number of key pairs used by the network is $a$, and each node possesses $b$ private keys. If $x$ nodes are captured, then $k_c(x) = a - (a - b) \left( \frac{(a-b)}{a} \right)^{x-1}$ expected number of keys will be disclosed. Then $\frac{C(\{k_c(x)\}, b)}{a}$ percentage of the node will be compromised. Here, we take the floor of $k_c(x)$, because in many cases $k_c(x)$ is not an integer.

Proof: Assume no bias when nodes choose private keys from the key pool. When the fist node is captured, $b$ keys are disclosed and $k_c(1) = b$. If the $i$-th node is captured, $b \times \frac{a - k_c(i-1)}{a}$ new keys will be revealed. When $i$-th node is captured, totally $k_c(i)$ keys are disclosed:

$$k_c(i) = k_c(i-1) + b \times \frac{a - k_c(i-1)}{a}$$

Equation (4) implies $k_c(i) = \frac{a - b}{a} k_c(i-1) + b$, where $k_c(1) = b$. Let $y_i = k_c(i) - a$. Replace the $k_c(i)$ with $y_i + a$, then $y_i = \frac{a - b}{a} y_{i-1}$ and $y_1 = b - a$. We get $y_i = (\frac{a - b}{a})^{i-1} y_1$.

Therefore, $k_i = a - (a - b) \left( \frac{a-b}{a} \right)^{i-1}$. If all a node picks up all its $b$ keys from the set of disclosed keys, then the node is compromised. The total number of such nodes is $C(\{k_c(x)\}, b)$, and the total number of possible choices of key selection is $C(a, b)$. The percentage is thus obtained.