Abstract

In this paper, we study challenges and possible solutions to cryptographic key management in mission-critical networks. Existing symmetric key cryptography mechanisms do not scale well when supporting end-to-end secure and private communication. Existing public key cryptography mechanisms in self-organized ad hoc networks cannot resist a Sybil attack, where a single malicious node presents multiple identities to control a substantial fraction of a network. In this paper, we present a new paradigm of public key cryptography based on combinatorial design, called SMOCK, where nodes combine more than one key to encrypt and decrypt each message. Our key allocation scheme guarantees that a set of keys held by one user is not a subset of keys held by any other user. We show that our proposed method offers efficiency in memory usage, control data exchange, as well as controllable resilience against node captures.

1 Introduction

There are emerging needs of collaboration in mission-critical applications, including battlefield communication, emergency rescue operations, and disaster recovery. In such application scenarios, mobile nodes are dispatched into incident areas and thus form Incident Area Networks (IANs) to handle incidents. These IAN form together a mission-critical network and are established on demand and in ad hoc manner. It means that mobile nodes may leave one IAN and join another one. Within the mission-critical network, inter-operability among organizations is necessary, since different IANs may belong to different organizations. To facilitate inter-operability as well as protect security and privacy, secure communication is in great demand. To safely exchange messages among legitimate users and block unauthorized users from accessing network resources, we can deploy cryptographic keys to enforce secure communications. However, the cryptographic key management is challenging due to following characteristics of the mission critical networks:

- The number of ad hoc wireless devices deployed at an incident scene depends on specific nature of the incident. Today, it is typically smaller than 1000, but may be larger in the future. Hence, key management for such a network must be scalable.
- Besides link level security, privacy (end-to-end security) is a very important issue, and this is why end-to-end secure channel is desired.
- More legitimated nodes may join the network later after some mission-critical nodes are deployed in the field.
- Nodes in mission-critical networks are mobile. Mobility introduces difficulties in trust management.
1.1 Cryptographic Systems

Current solutions to key management in wireless ad hoc networks rely on two cryptographical algorithms: symmetric cryptography and public key cryptography. Symmetric key techniques are attractive due to their efficiency in computation and power consumption. In symmetric key pre-distribution design for wireless sensor networks, our goal is to use small amount of storage to achieve good secure connectivity and good resilience to node captures. However, current design of key management for wireless sensor networks does not guarantee that each pair of nodes share a unique secret key. Therefore, privacy of end-to-end secure communications cannot be preserved. In mission-critical networks, privacy is an important issue. We assume that communications between two nodes must not be overheard by any other node, even if the node is the insider of mission-critical network.

Recently, Elliptic Curve Cryptography (ECC) has been emerging as an attractive PKC scheme for mobile/wireless environments [10] [9] [18]. In public key cryptography, any two nodes can establish a secure channel between them without necessarily carrying pre-distributed keys. However, if nodes do not carry pre-distributed keys, one or more trusted certificate authorities (CAs) are needed. Otherwise, the network cannot resist Sybil attacks [19], where a single malicious node presents multiple identities to control a substantial fraction of the network, thereby undermining the system performance. In mission-critical networks, authentication process by CAs is very costly in terms of wireless communication overhead. Further, CAs may suffer from failures and attacks and stop functioning, as a result, the whole network cannot operate smoothly.

Therefore, in mission-critical networks, we should issue public keys to all of the nodes beforehand. Although operating in an ad hoc manner, mission-critical networks allow pre-distribution of security keys. Before being dispatched to an incident area, nodes in the mission-critical network are able to communicate securely with trusted authentication server in their domain center. However, once the nodes are dispersed to the field, authentication server (domain) loses control of these nodes, and the nodes cannot trust each other without the protection of secure keys. Figure 1 shows an example scenario of mission-critical networks.

![Diagram](image)

**Figure 1**: Two authentication servers (police department and emergency medical service (EMS)) maintain the same private and public key pool through a secure connection. Before deployment, their authentication servers pre-distribute keys to devices. After devices are dispatched into the incident areas, it is costly and unsafe to communicate with authentication servers. So all the devices authenticate messages according to the pre-distributed keys.

Since an authentication server is only available at network initialization phase and every time a node joins, it is desired that nodes contact the authentication server in order to receive public key or a symmetric key, before joining the network. Assume that network size is $n$. In order to build a secret communication channel between any pair of nodes, a conventional key pre-distribution scheme requires each node to have $n-1$ keys. Moreover, each time a new node joins the network, $n-1$ distinct keys must be distributed respectively to $n-1$ existing nodes. Hence, storage requirement at each node and communication overhead for secret key distribution to a new node are both of $O(n)$. 


1.2 Design Goal

We aim to design a method for cryptographic key management, which is scalable. Specifically, we require that storage requirement at each node and communication overhead for key distribution to a new node are significantly less than $O(n)$. However, we allow computation requirement to be somewhat relaxed as long as processing delay remains acceptable. Such relaxation is reasonable because the length of continuous use of a device carried by a first responder is naturally constrained by the amount of time the first responder is able to spend in an incident scene management operation, and the device may be recharged or have its battery replaced whenever the first responder is recalled for duty rotation. Therefore, our design for a key management scheme needs to have following features:

1. **Small overhead**: The mission-critical networks are formed on demand and in ad hoc manner, and wireless link capacity is limited. Communication overhead for authentication should be very small.

2. **Fault tolerance**: Mobile nodes in mission-critical network are prone to faults, so PKC protocols should be tolerant to a fraction of node failures.

3. **Resilience**: Mobile wireless nodes are more vulnerable to various attacks due to the lack of physical protection of wireless links. PKC protocols should operate securely even when a small amount of nodes is compromised.

4. **Availability**: Connectivity is not guaranteed in a mission-critical network. However, if any pair of nodes is connected, secure communication should be available.

5. **Small Footprint**: Due to the limitation of memory on a mobile node, a small amount of memory should be consumed to achieve (1), (2), (3) and (4).

1.3 Basic Idea

Our solution is inspired by the use of multiple keys to open a door to a vault. In SMOCK, a sender uses multiple keys to encrypt a message and a receiver needs to use multiple keys to decrypt the message. We use public key cryptography as follows: Each node possess a unique combination of private keys, and knows all the public keys. The private key combination pattern is unambiguously associated with the node ID. Now if a sender $A$ wants to send a message to a 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 private keys owned by $B$. The SMOCK scheme has been verified to be very efficient and satisfy the overall design goals via simulation results.

1.4 Paper Organization

The paper is organized as follows: In Section 2, we summarize related work in key management. In Section 3, we describe the background and problem description. Section 4 provides detailed key allocation algorithms. Section 5 illustrates how the proposed key management scheme resists against attacks. Section 6 gives detailed protocols for secure communication and bootstrapping when new nodes are deployed. Section 7 assesses the proposed scheme. Finally, Section 8 provides some concluding remarks.

2 Related Work

Symmetric key techniques have been proposed in [1] [2] [3] [4] [5] [6] [7] [8]. 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 secret key design is to use small memory footprint to establish as many secure communication pairs as possible and achieve the highest level of resilience. However, these techniques are not storage efficient for a large-scale network.

Eschenauer and Gligor propose a random key pre-distribution scheme [4] to address the storage limitation problem of the symmetric key allocation. In the random key pre-distribution scheme, each node selects a subset of random
keys from a pool of keys before deployment. The probability that any pair of nodes possesses at least one common key is \( p \), thus with \( p \) probability two nodes can share secret.

To increase resilience of a network against node capture, Chan, Perrig and Song extend the random key pre-distribution scheme to use \( q \)-composite keys to establish a secure link [2], where \( q \) \((q > 1)\) common keys are needed instead of just one. In random key scheme, two immediate neighbors are connected by a secure link with probability \( p \), and there is always a chance that the graph may not be fully connected, and the chances are increasing as \( q \) increases. While detecting the disconnection, the network can increase transmission range by increasing transmission power, and thus introduce more interference. Another limitation of random key schemes is communication overhead during key set up phase after deployment.

In a master key based protocol [11] by Lai, Kim and Verbauwhede, a single master key is pre-distributed to all nodes in a network. A pair of nodes use the master key to establish a session key. Each node uses one unit of memory to store the master key, and it is very memory efficient. However, resilience of the master key scheme is poor since once the master key is disclosed, all links are compromised.

Camtepe and Yener propose a combinatorial design of key distribution of symmetric keys [1], where \( m \) is a design parameter. The scheme supports \((m^2 + m + 1)\) nodes in the network and the key-pool size is \((m^2 + m + 1)\). Each node carries \( m + 1 \) keys and every pair of nodes has exactly one key in common. Therefore, communications among network nodes are secure. When one node is captured, with the probability of \( \frac{1}{m} \), a link in the network will be compromised. The limitation of this scheme is that it does not apply to arbitrary number of nodes in the network.

In both random key pre-distribution and combinatorial design scheme, a key is shared by several nodes. Thus the communication of any pair of nodes may be overheard by some other nodes which hold the same key. Therefore, privacy cannot be preserved under such mechanisms. In mission-critical networks, privacy is crucial as well.

PKC approaches were originally targeted at the Internet [12]. In order to tailor PKC approaches to ad hoc networks, Zhou and Haas propose a distributed public-key management scheme for ad hoc networks [13], 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 fully distributed scheme [14], where every node carries a share of the private key of the service. This scheme increases availability of authentication. But on the other hand, it increases communication overhead for authentication. [10] shows that it is practical to use PKC for sensor networks, where computational power of each device is low. Capkun, Buttyan, and Hubaux propose a self-organized public key management system [15], 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.

Montenegro and Castelluccia propose a scheme to bind node IDs to public keys in [16], where the public key is hashed, and the hash value is used as part of the IP address of the node. Therefore, the certificates to bind the node ID to its public key is not necessary.

In summary, the PKC mechanisms are usually memory efficient and can achieve excellent resilience, however, they are vulnerable to Sybil attacks. In this paper, we utilize the combinatorial design of PKC to achieve secure communication and protect privacy between each pair of nodes.

### 3 Problem Definition

Assume a group of people who want to exchange correspondence securely between each pair of them. There is a set of public keys available to all. But each person keeps a different subset of private keys. The key pool of such a system \( K \) consists of a set of private-public key pairs. Each key pair consists of two mathematical related keys. Let \( K_{\text{priv}}^{\text{Alice}} \) denote a subset of private keys held by Alice, and \( K_{\text{pub}}^{\text{Alice}} \) represent the corresponding public key subset. If Bob wants to send his secret message to Alice, he needs to know \( K_{\text{pub}}^{\text{Alice}} \), where \( K_{\text{priv}}^{\text{Alice}} \not\in K_{\text{priv}}^{\text{anybody else}} \). Bob is able to pass the secret message to Alice, using the public keys \( K_{\text{pub}}^{\text{Alice}} \) to encrypt message. The message can only be opened by Alice, who has the private key set \( K_{\text{priv}}^{\text{Alice}} \), but others do not. In this scenario, we know that

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

A message is encrypted by a subset of public keys. The message can only be read when the message can be decrypted by the subset of private keys corresponding to the subset of public keys.

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

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>A Key pool: a set of public-private key pairs</td>
</tr>
<tr>
<td>$privateKey_{ij}$</td>
<td>j-th private key hold by user $i$, $j \in [1, b_i]$</td>
</tr>
<tr>
<td>$publicKey_{ij}$</td>
<td>j-th public key hold by user $i$, $j \in [1, b_i]$</td>
</tr>
<tr>
<td>$K^\text{priv}_i$</td>
<td>A set of private keys held by user $i$, $K^\text{priv}<em>i = {privateKey</em>{ij}</td>
</tr>
<tr>
<td>$K^\text{pub}_i$</td>
<td>A set of public keys corresponding to $K^\text{priv}_i$</td>
</tr>
<tr>
<td>$K_i$</td>
<td>A set of public-private key pairs held by user $i$, $K_i = {(k^\text{priv}, k^\text{pub})</td>
</tr>
<tr>
<td>$M_k(x)$</td>
<td>Memory size for key storage</td>
</tr>
<tr>
<td>$k_c(x)$</td>
<td>Expected number of disclosed keys when $x$ nodes are captured</td>
</tr>
<tr>
<td>$k_v(x)$</td>
<td>Maximum number of disclosed keys when $x$ nodes are captured</td>
</tr>
<tr>
<td>$V_x(a, b)$</td>
<td>Vulnerability metrics as $x$ nodes are captured.</td>
</tr>
<tr>
<td>$C(a, b)$</td>
<td>Abbreviation of $\binom{a}{b}$</td>
</tr>
<tr>
<td>$n$</td>
<td>Total number of nodes in the network, $n =</td>
</tr>
</tbody>
</table>

### 3.1 Definitions

**Definition 1:** A public-private key pair is defined as $(k^\text{pub}, k^\text{priv})$. The symbol $K^\text{priv}_v$ stands for a set of private keys held by node $v$. Then the key pool $K = \{(k^\text{pub}, k^\text{priv})| k^\text{priv} \in K^\text{priv}_v, \forall v \in V\}$. $a = |K|$ represents the number of distinct keys.

**Definition 2:** A key allocation $KA: 2^K \to V$, maps the key pairs in $K$ to a set of nodes in $V$, so that $v \in V$ is assigned a subset of key pairs $K_i$ ($K_i \subset K$). 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 $k^\text{priv}_i \not\subseteq k^\text{priv}_j$) and $K_j \not\subseteq K_i$ (the same as $k^\text{priv}_j \not\subseteq k^\text{priv}_i$), 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 $i$, $K_i$ and the key assignment to user $j$, $K_j$ conflict, if either $K_i \subseteq K_j$ or $K_j \subseteq K_i$. 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 between $n$ people, we need to have enough public-private key pairs. On the other hand, similar to a vault system, we want to use a small number of locks and distribute a small number copies of keys to each person for efficient key management. Generally, we desire the key management to be memory efficient for key storage, communication efficient during authentication, computationally efficient during encryption and decryption, and resilient under node captures. Therefore, we define multiple objectives of the key allocation mechanism as:
Therefore, $|K_i|$ keys need to be stored. Note each node stores all the public keys before deployment, but only store a small subset or
following algorithm helps to determine a communications.

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

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

In SMOCK, each user needs only to carry $b$ private keys and $a$ public keys, wherein $(a + b) << n$. The price to pay for our scheme is controllable compromise in resiliency against node captures. Clearly if a node is captured, all its keys are compromised, regardless of the number of private keys it carries. However, in our scheme, the capture of $x$ nodes compromises $C(k_v(x), b)$ distinct key-sets on the average, and up to $C(k_v(x), b)$ distinct key-sets in the worst case. It follows that the vulnerability metric, $V_x(a, b)$ is $\frac{C(k_v(x), b)}{C(a, b)}$ on the average or $\frac{C(k_v(x), b)}{C(a, b)}$ in the worst case. Clearly, $C(k_v(x), b)$ 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 as benchmark resilience.

4 Key Allocation Algorithm

Our key management scheme, SMOCK, will use the isometric key allocation algorithms to achieve the objectives outlined in Section 3.2. The reason is that the isometric scheme achieves better performance than non-isometric allocations as shown in Appendix (see Proposition 3). In this section we show: (1) Under isometric key allocation, determine the size of key pool $a$ and $b$; (2) Allocate distinct private key sets to users to build secure channel between each pair of users. We first specify heuristic algorithms to obtain a near optimum key allocation solution in terms of both Objective 1 and Objective 2. We then make some adjustment to the solution to satisfy the resilience requirement in Objective 3.

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 $b = 1$, 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 Memory Efficiency Given a network of size $n$, we need to find a key pool $K$ and a key allocation $KA$ to achieve

\[
\begin{align*}
\min & \quad |K| + \max_{i \in V} b_i \\
\text{s.t.} & \quad K_i \not\subseteq K_j \text{ and } K_i \not\supseteq K_j \quad \forall i \neq j
\end{align*}
\]

where $b_i = |K_i| = |K_i^{\text{priv}}|$ is the total number of private keys stored at node $i$, $|K| = a$ is the total number of public keys need to be stored. Note each node stores all the public keys before deployment, but only store a small subset or private keys $K_i^{\text{priv}}$ for user $i$. If a user is assigned a key pair $(k_{\text{pub}}, k_{\text{priv}})$, then the user holds the private key $k_{\text{priv}}$. Therefore, $|K| + b_i$ 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

\[
\begin{align*}
\min & \quad \max_{i \in V} b_i \\
\text{s.t.} & \quad K_i \not\subseteq K_j \text{ and } K_i \not\supseteq K_j \quad \forall i \neq j
\end{align*}
\]

Objective 3 Resilience Requirement Under isometric key allocation scheme, we denote a vulnerability metric by $V_x(a, b)$, which is the percentage of communications being compromised when $x$ nodes are captured. To achieve desired resilience under capture of $x$ nodes, we define

\[
4.1.1 \quad \text{Optimization of design objectives}
\]

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 $b = 1$, 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 $n$. 

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 $b = 1$, 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 $n$. 

\[
\begin{align*}
\min & \quad |K| + \max_{i \in V} b_i \\
\text{s.t.} & \quad K_i \not\subseteq K_j \text{ and } K_i \not\supseteq K_j \quad \forall i \neq j
\end{align*}
\]
Objective 1 requires \( \frac{a}{n} \) to be small for key storage efficiency. Meanwhile Objective 3 requires \( \frac{b}{5} \) 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 \( \{ l = l + 1; \) \)
   \( a = l, \ b = \lfloor \frac{l}{2} \rfloor \);
(2) While \( (C(a, b - 1) > n) \)
   do \( \{ b = b - 1; \) \)
(3) While \( (C(a + 1, b - 1) > n) \)
   do \( \{ a = a + 1, \ b = b - 1; \) \)
(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 \( \frac{b}{5} \) 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.

4.1.2 Meeting key storage constraint

Total memory slots for key storage is often limited to \( M \), 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 the left hand side of Objective 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 determine \( a \) and \( b \). Then, we randomly assign \( b \) private keys to each node. A single key should be assigned to at most \( \frac{a}{b} C(a, b) \) nodes; otherwise, we cannot get a valid key allocation. The key assignment should make \( \mathcal{K}_i \not\subseteq \mathcal{K}_j \) 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 \( C(a, b) \). Note that for a very large network size, if we do not consider the resilience requirement, \( a \) and \( b \) can be very small. E.g., \( a = 20, \ b = 4 \), the network size can be as large as 4845.
5 Resistance Against Attacks

5.1 Against Eavesdropping

Eavesdropping is a passive attack, where an unauthorized recipient intercepts the message. To prevent eavesdropping and protect privacy, when a node $v_i$ sends a message $M_{ij}$ to $v_j$, the node first truncates $M_{ij}$ into $b$ pieces, $M_{ij}(1), M_{ij}(2), \ldots, M_{ij}(b)$ and encrypts the message as follows:

$$
S_{ij1} = E(publicKey_j1, M_{ij}(1))
$$

$$
S_{ij2} = E(publicKey_j2, M_{ij}(2) \oplus M_{ij}(1))
$$

$$
\ldots
$$

$$
S_{ijb} = E(publicKey_{jb}, M_{ij}(b) \oplus M_{ij}(b-1))
$$

(4)

Since a receiver $v_j$ knows the private keys in $K_{pr}^j$, $v_i$ can decrypt the message as follows:

$$
M_{ij}(1) = D(privateKey_j1, S_{ij1})
$$

$$
M_{ij}(2) = D(privateKey_j2, S_{ij2}) \oplus M_{ij}(1)
$$

$$
\ldots
$$

$$
M_{ij}(b) = D(privateKey_{jb}, S_{ijb}) \oplus M_{ij}(b-1)
$$

(5)

5.2 Avoid Spoofing

If node $v_i$ sends a message and wishes to assure the receiver that nobody fakes the message or the sender’s ID, it encrypts the message as follows:

$$
S_{ij1} = E(privateKey_{i1}, M_{ij}(1))
$$

$$
S_{ij2} = E(privateKey_{i2}, M_{ij}(2) \oplus M_{ij}(1))
$$

$$
\ldots
$$

$$
S_{ijb} = E(privateKey_{ib}, M_{ij}(b) \oplus M_{ij}(b-1))
$$

(6)

At the receiver side, node $v_j$ recovers the message as follows:

$$
M_{ij}(1) = D(publicKey_{i1}, S_{ij1})
$$

$$
M_{ij}(2) = D(publicKey_{i2}, S_{ij2}) \oplus M_{ij}(1)
$$

$$
\ldots
$$

$$
M_{ij}(b) = D(publicKey_{ib}, S_{ijb}) \oplus M_{ij}(b-1)
$$

(7)

5.3 Prevent DoS Attack

A malicious node or an inattentive node from outside of the mission-critical network can launch DoS attack on data traffic by injecting a significant amount of data traffic into the network to clog the network. Without protection mechanism, legitimate user packets will be dropped along with malicious ones as the result of congestion. In this case, signature of a legitimate sender is required for routing request message. In traditional public key cryptography (PKC), a node in the self-organizing network may not possess the public key of the sender, thus need communicate with its neighbors to verify the signature of the sender. In the SMOCK framework, since the key pool is usually small, each intermediate node keeps all the public keys locally, thus can easily check whether the signature from a sender is correct alone.

5.4 Facilitate Anonymous Routing

It is straightforward for SMOCK to render anonymous routing very convenient. For example, if a sender knows all intermediate nodes on a route to a destination, it has an option to encrypt all packet headers so that each intermediate node can only learn about its next hop, but not other nodes on the route.
5.5 Resist Sybil Attack

With pre-distributed keys, a malicious node cannot initiate a Sybil attack, since the user cannot circulate fake keys in the network. Secure communication requires proper keys from both sender and receiver, while all the public keys are known by each of the users. Since the total number of public keys is small, it is feasible for each user to carry all of them. In our scheme, secure communication is not based on secure key exchanging. Therefore, the attacker cannot circulate fake keys to launch a Sybil attack.

6 Secure Communication Protocols

In this section we specify detailed protocols used for initialization, communication, and bootstrapping. 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).

6.1 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) represents a subset of keys the node possesses. In future communication, if two nodes want to exchange secure message, each needs to know the ID of the other. From the ID, a node can tell which private keys the other has, and it can encrypt the message by 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 $0, 1, 2, \ldots$. 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 2(a). 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.

![ID field of node $j$](a) ID field of node $j$

![Secure communication between Alice and Bob](b) Secure communication between Alice and Bob

Figure 2: Secure communication protocols

6.2 Secure Communication

Figure 2(b) shows a procedure of secure communication between Alice and Bob, where Alice and Bob establish a secure communication channel from Alice to Bob. 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 public 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, and any malicious user who steals Bob’s ID cannot decrypt the encrypted message.

6.3 Bootstrapping to Accommodate New Nodes

Assume that a previously deployed network contains $a$ public keys and each user possesses $b$ private keys. Suppose we desire to deploy more nodes in the field. If the total number of deployed nodes (including previously
deployed nodes and new nodes) is smaller than $C(a, b)$, then no bootstrapping is necessary. The newly deployed nodes can choose unused combinations of private keys from the existing key pool.

If network size is larger than $C(a, b)$ after incremental deployment, then the system needs to generate more key pairs, say $a'$ new key pairs, and assign $b$ private keys to the additional nodes before their deployment, where $b$ private keys must include at least one key from the newly created key pairs. This implies that bootstrapping information for additional nodes must be present. We fix the number of private keys possessed by each node. After new nodes join the network, they need to broadcast the newly generated public keys and the number $a'$ to their neighbors. Each node in the network only needs to broadcast one such bootstrapping message.

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 $a$ doesn’t violate the resilience bound $P$ given in Objective 3.

To prevent unauthorized nodes from establishing communication channels with legitimate nodes and thus gain entry into the network, bootstrapping information should be encrypted. Since the number of previously deployed nodes is smaller than $C(a, b)$, there must exist at least one newly deployed node that can choose all its $b$ private keys from the existing $a$ key pairs. Therefore this node should initiate the broadcasting of the newly added $a'$ public keys and encrypt the message by its private keys according to section 5.2. When the previous deployed nodes receive such information, they encrypt the information by their private keys and broadcast it to their neighbors. Since all nodes know the previously generated public keys, such message will be verified and forwarded. Therefore, the whole network will be aware that some new legitimate nodes are joining the network, and new public keys are generated. The resulting total number of key pairs in the network is $a + a'$ and the ID field takes a larger size as $b \times \lceil \log_2(a + a') \rceil$. Each network node should reconfigure its ID as shown in section 6.1.

7 Evaluations

7.1 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 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.

![Graph showing minimal number of keys needed](image)

Figure 3: The minimal number of keys needed

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

7.2 Communication Overhead

It is not necessary for SMOCK to perform key setup or verification in the field. Therefore, SMOCK has virtually no overhead for authentication after the deployment.

7.3 Resilience to Node Failure

The proposed scheme is able to tolerate a fraction of node failures. Node failures in a network do not affect the secure communication between any other pair of nodes as long as the network remains connected, since each node keeps its secret keys for communication locally.

7.4 Resilience to Node Capture

7.4.1 Average case analysis

Capture 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, capture of multiple nodes may compromise a set of other nodes. Assume $x$ nodes are captured and $k_c(x)$ is the expected number of keys disclosed if $x$ nodes are captured. As Lemma 3 in Appendix shows, $k_c(x) = a - (a - b) \left(\frac{a-b}{a}\right)^{x-1}$. Then $\frac{C(k_c(x), b)}{C(a, b)}$ percentage of the node 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 captured.

7.4.2 Worst case analysis

For mission-critical applications, it may be important to consider resilience against the worst case where each newly captured node releases $b$ new keys to the adversary. If we define $k_c(x)$ as the number of keys disclosed by the capture of $x$ nodes, then in the worst case, $k_c(x) = \min(xb, a)$, 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 $\text{Prob}\{a \text{ key set is compromised} \mid \text{the key set is allocated}\}$. Since the events “a key set is compromised” and “a key set is allocated” are independent, then the worst case probability

Figure 4: Trade-off between storage space and computational complexity
Figure 5: The percentage of compromised nodes with the node capture

is \( \text{Prob(a key set is compromised)} \). Therefore, given \( a \) and \( b \), in worst case, the capture of \( x \) nodes results in 
\[
\frac{C(k_c(x), b)}{C(a, b)} = \text{percent of the communication compromises, where } n \text{ is the network size. Figure 5(b) shows the worst case percentage of the compromised nodes, where we can see that the capture of } \lceil \frac{a}{b} \rceil \text{ nodes can compromise the whole network in the worst case. However, the capture of } \lceil \frac{a}{2b} \rceil \text{ nodes only compromise a small ratio of the network.}
\]

### 7.4.3 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 is 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 capture of any single node cannot compromise any other node in the network, and capture of multiple nodes may disclose information to the adversary to compromise more than the number of captured nodes. The capture of multiple nodes will be more expensive for the adversary than the capture 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:
\[
\frac{C(k_c(x), b)}{C(a, b)} < 20\%.
\]
When 20 or fewer nodes are captured, we require that at most 20\% of the secure channels are compromised. According to Algorithm 1, the minimum memory slots needed to fulfill such resilience requirement is 70.

Figure 6 compares the total number of keys needed to achieve \( V_x(a, b) < x/n \) under \textit{SMOCK} with the conventional public key scheme. We assume that only a small subset of nodes may be captured. Figure 6(a) and Figure 6(b) show that when \textit{SMOCK} achieves benchmark resilience at \( x = 20 \), it performs better resilience for \( x \leq 20 \), but requires 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 \( x = 100 \). For applications with a high resilience requirement, we recommend using \( x/n \) as the resilience bound in Objective 3.

### 7.5 Ability to Expand Network

The protocol describing the bootstrapping procedure when new nodes are deployed to an existing network is specified in section 6.3. New nodes should let the previously deployed nodes know their public keys. Since a small number of keys can support a large network size, the communication overhead for broadcasting the public keys during bootstrapping is moderate. On the other hand, since at least one newly deployed node chooses all its private keys from the known key pairs of the existing network, it can encrypt and broadcast the newly generated public keys by its own signature. Thus the network has the ability to distinguish between legitimate nodes and malicious nodes which try to gain access to the network.

\(^1\)The node capture detection and key revocation is beyond the scope of this paper.
8 Concluding Remarks

We propose a key predistribution scheme, which requires significantly less than $O(n)$ key storage overhead and communication overhead in an wireless ad hoc network with $n$ nodes. The scheme also achieves controllable resilience against node captures. The characteristics of mission-critical networks imply that the proposed scheme is promising in the following aspects.

- Scalability and the ability to dynamically deploy additional nodes.
- Resistance against malicious nodes, who pretend to be legitimate nodes (Sybil Attack)
- Resilience to node capture. By our analysis, in order to get better resilience, $b$ cannot be large. Usually $b$ is an integer between 2 to 4.
- Protection of communication privacy (end-to-end security), such that the talk between any pair of nodes doesn’t disclose secret to other nodes.

We further address the following issues to conclude our paper.

8.1 Computational Complexity

In SMOCK, the computational complexity is $b$ times that of a widely known traditional PKC mechanism. However, if we use Diffie-Hellman protocol on top of SMOCK to establish session keys, as well as caching of the session keys, we can further reduce the computational overhead. As explained in Section 1.2, for key management in a mission-critical ad hoc wireless network, it is reasonable for computation requirement to be somewhat relaxed as long as processing delay remains acceptable. Furthermore, the idea of using non-symmetric keys to trigger symmetric keys excludes the computational complexity as a major concern in our design.

8.2 Key Length in SMOCK

In this paper, we have discussed the needed storage space in terms of the number of keys. Strictly speaking, we should also take into consideration the size of keys used in different cryptography techniques for fair evaluation of SMOCK. Given $n$ nodes, the storage required in a conventional symmetric key technique is $\eta_s n$, where $\eta_s$ denotes length of a symmetric key. On the other hand, SMOCK, requires each node to carry $a$ public keys and $b$ private keys, such that the total storage required is $(\eta_a a + \eta_b b)$ (in most cases $\eta_a = \eta_b = \eta$). To demonstrate an advantage of SMOCK over conventional techniques, we need to show not only $a + b << n$, but also $\eta(a + b) << \eta_s n$.

[20] illustrates the information regarding symmetric and RSA asymmetric key lengths with similar resistance to brute force attacks. Accordingly, to achieve equivalent level of security, we should select much longer RSA asymmetric keys than symmetric keys. However, [21] [22] show that Elliptic Curve Cryptography (ECC) can...
offer equivalent security with substantially smaller key sizes. For example, key size of 160 bits in ECC achieves equivalent security to 80 bit symmetric key scheme and 1024 bit RSA, and 521 bit ECC is equivalent to 256 bit symmetric key scheme and 15360 bit RSA. With roughly $\eta = 2\eta_s$ under ECC algorithm, we have $\eta(a + b) << \eta_s n$. Therefore, we conclude that SMOCK achieves an excellent memory efficiency.

References


APPENDIX

Lemma 1: In a valid non-isometric key allocation $KA$, if node $i$ is assigned the smallest key pair set, and node $v$ is assigned the largest key pair set, then there exists a valid key allocation to increase $b_i$ by 1 with $b_v$ 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 $KA'$, $\mathcal{K}_i \not\subseteq \mathcal{K}_j$ must hold, if it holds in $KA$. In $KA'$, $\mathcal{K}_j \not\subseteq \mathcal{K}_i$ must hold also, otherwise, $\mathcal{K}_j = \mathcal{K}_i$ in $KA'$, which implies $\mathcal{K}_i \subset \mathcal{K}_j$ in $KA$, thus it contradicts the fact that $KA$ 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 exist 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 $\mathcal{K}_i$ 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 $\mathcal{K}_i$ by adding any key pair to $\mathcal{K}_i$, and can always find $j \in J$ and $\mathcal{K}_j \subset \mathcal{K}_i$. In this case, we can replace the original $\mathcal{K}_j$ with $\mathcal{K}_i$ and keep the original $\mathcal{K}_i$. Such replacement is equivalent to adding a new key pair to $\mathcal{K}_j$’s. So far, the key allocation is invalid since $\mathcal{K}_i$ still contradicts with a $\mathcal{K}_j$, $j \in J$. We can create $(a - b_i - 1)$ potential assignment options by adding one key pair to each $\mathcal{K}_j$, $j \in J$. Among all these options, we have totally $\sum_{k=1}^{a-b_i-1} k$ distinct new key assignment options, since any two users create a common key assignment option. We have at most $a - b_i$ replacements of $\mathcal{K}_j$. Since $\sum_{k=1}^{a-b_i-1} k > a - b_i$, so after $a - b_i$ replacement, we still can find an option to replace the original $\mathcal{K}_i$ and create a valid key allocation. Thus, $b_j$ 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 $b_v$ unchanged.

Proposition 1: Given $\mathcal{K}$ 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 $\mathcal{K}_i = \{k_{i1}, k_{i2}, \ldots, k_{ib_i}\}$, then the number of keys kept by node $i$ is denoted as $|\mathcal{K}_i| = b_i$. 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 $b_l > b_t$. 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 \( b_l > b_{l+1} \), then we have chance to further optimize Objective 1 and Objective 2 by making \( b_l = b_{l+1} \).

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 \( KA \).

**Lemma 2:** Given a set of key pairs \( K \), wherein \( |K| = a \). The network size is \( n \). Therefore, we need at least \( n \) key sets. If \( C(a, b) < n \leq C(a, b + 1) \), for \( b < \left\lfloor \frac{a}{2} \right\rfloor - 1 \), an isometric allocation utilizing only key sets of a fixed size \( b + 1 \) minimizes Objective 2 and hence Objective 1 among all of the isometric allocations.

**Proof:** For isometric key allocation, the proof is based on the following fact that for \( a \geq b \geq 1 \), \( C(a, b) \) is increasing as \( 1 \leq b \leq \left\lfloor \frac{a}{2} \right\rfloor \). Otherwise, \( C(a, b) \) is decreasing as \( \left\lfloor \frac{a}{2} \right\rfloor \leq b \leq a \). When \( a \) is odd, \( C(a, \left\lfloor \frac{a}{2} \right\rfloor) = C(a, \left\lceil \frac{a}{2} \right\rceil) \).

**Proposition 2:** Given \( a \) and \( u \leq \left\lfloor \frac{a}{2} \right\rfloor \), then there exists an isometric key allocation to support \( n = C(a, u) \) nodes, and each of the node is assigned a key set of length \( u \). For other key allocations to support \( n \) nodes, the maximum length of assigned key set is larger than \( u \).

**Proof:** Given \( a \), we define Group \( b \) to be a set of all \( C(a, b) \) key sets of size \( b \). When \( n = C(a, u) \), the isometric allocation using all key sets in Group \( u \) is optimal in accordance with Lemma 2. For any other allocations, at least one key set is selected from higher groups or lower groups. Without using key set from higher groups, it is not possible to replace a key sets in group \( u \) with one from lower groups, since any key set from those groups will be a subset of at least one key set in group \( u \).

Also we can conclude that given \( a \), when \( n > C(a, \left\lfloor \frac{a}{2} \right\rfloor) \), there exists no valid allocation for \( n \) nodes.

**Proposition 3:** Isometric allocation of keys performs better than non-isometric allocation.

**Proof:** Proposition 3 is intuitive, based on proposition 1 and proposition 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.

**Lemma 3:** 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 \( k_x = a - (a - b) \left( \frac{a-b}{a} \right)^{x-1} \) expected number of keys will be disclosed. Then \( \frac{C(\lfloor k_c(x) \rfloor, b)}{C(a, b)} \) percentage of the node will be compromised.

**Proof:** Assume no bias when nodes choose private keys from the key pool. When the first 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 (8) 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 \( y_i = \left( \frac{a-b}{a} \right)^{i-1} y_1 \). 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 \( C(\lfloor k_c(x) \rfloor, b) \), and the total number of possible choices of key selection is \( C(a, b) \). The percentage is thus obtained.