A Distributed Public Key Caching Scheme in Large Wireless Networks

Yuan Kong Jing Deng Stephen R. Tate
Department of Computer Science, University of North Carolina at Greensboro, Greensboro, NC, USA
Email: {y_kong2, jing.deng, srtate}@uncg.edu

***Abstract*— When asymmetric cryptography techniques are used in wireless networks, the public keys of the nodes need to be widely available and signed by a Certificate Authority (CA). However, the existence of a single CA in large wireless networks such as mobile ad hoc networks and wireless sensor networks can lead the hotspot problem and become a security weakness. In this work, we propose a distributed technique to cache the public keys on regular nodes. Due to the limited memory size that each node is allowed to dedicate for key caching, only some keys can be cached. In our proposed technique, each node caches the public keys of a mix of local and remote nodes. Access to the public keys of other nodes is possible based on a chain of trust. Multiple copies of public keys from different chains of trusted nodes provide fault tolerance. We explain our technique in detail and investigate its salient features in this work. An interesting observation is the need to balance caching public keys of local nodes and remote nodes.**

I. Introduction

Large wireless networks are expected to play an increasingly important role to provide networked information. Examples include mobile ad hoc networks (MANETs) and wireless sensor networks (WSNs). WSNs are composed by a large number of sensors which can communicate with each other and monitor environmental conditions cooperatively. In such networks, each node plays the same role and supports multi-hop routing. There is no specific server or central control.

With more and more information delivered on these wireless networks, security becomes a critical issue. The asymmetric cryptography scheme can be used to provide information security. Using the asymmetric key scheme, each node in the network has a pair of keys: public key and private key. The public keys should be known by all nodes in the network; the private key should only be held by the node itself. For information confidentiality, a node uses the public key (of a receiver) to encrypt the message. This encrypted message can only be decrypted by the intended receiver who holds the matching private key. For information authentication, a node uses its own private key to sign the message. In order to authenticate the message, the receiver needs the public key of the sender.

However, the availability of these public keys can be an issue in large wireless networks. Usually, there is a Certificate Authority (CA) that will issue certificates for every node. Each certificate, signed by the CA, contains a public key and the identifier of a node. In large wireless networks, the existence of such a CA can become a security weakness. For example, since the CA is known to the entire network, the adversary can attack it with all its resources. The traffic toward the CA can be mis-routed with the use of worm-hole attacks [1] or black-hole attacks [2]. Jamming attacks can be launched by the adversary to blackout all wireless communications in the CA's neighborhood.

In this work, we investigate a wireless network where every node can serve as a CA, or a Peer CA (PCA). Since the certificate from one PCA can be unreliable (imagine a compromised node serving as a PCA), the certificates from a number of PCAs will confirm that the public key of a certain node is valid. This is similar to threshold cryptography [3] and distributed trust [4].

Wireless devices are usually resource-constrained, in terms of computational power, battery energy, and on-board memory space. For example, the prototype sensors in WSNs have about 4K bytes of memory, which needs to support several important tasks: data collection and on-board processing; system parameter storage; and key storage. The limited memory space that can be dedicated to key storage gives rise to the following problem: how should the keys be cached and updated in such large wireless networks?

In this work, we propose a distributed key caching scheme. In this scheme, each node caches the public keys of some other nodes in the network. We investigate our public key caching scheme through the availability of the public key copies and observe the existence of an optimal local/remote ratio in different network scenarios. We also design a key update strategy that will allow nodes to update their public key caches according to the optimum local/remote ratio. The update process balances the cache for local/remote nodes and will be employed by all nodes.

The paper is organized as follows: Section II describes recent related works. In Section III, our public key caching scheme is explained in detail. Simulations are performed to evaluate our scheme in Section IV. In Section V, we summarize our work and discuss future works.

II. Related Work

Our work is mostly related to distributed trust establishment and security protection. We discuss several related work in the following.

In [5], mobile certificate authorities (MOCAs) were established for heterogeneous mobile ad hoc networks (MANETs) where some nodes are more reliable and resourceful than others. These MOCAs share the responsibility of collectively

providing the CA functionality for the network, using threshold cryptography [3]. In the MOCA certification Protocol (MP), a client requiring certificate service broadcasts a Certificate Request (CREQ) message and waits for responses from at least k out of the n MOCAs. With such k responses, the certificate can be fully reconstructed and the certification process succeeds.

Similarly, in [6], a fully distributed trust model for MANET was introduced based on trust graph and threshold cryptography. In their model, users can issue public key certificates and authentication can be performed via certificate chains. To alleviate the adverse effect of malicious nodes in the network, threshold cryptography was used. Thus, a user needs to acquire k partial certificates for authentication.

In [7], a decentralized key management architecture was designed for WSNs. This architecture supports key deployment, key refreshment, and key establishment. Symmetric cryptography was used in the keying protocols in [7].

In the fully self-organized public-key management system presented by [8], all the users in a MANET can generate their own public/private key pairs. The users can also store, distribute, and revoke their public keys by themselves. Each node maintains a certificate repository using two ways: by communicating with its certificate graph neighbors; and by applying a repository construction algorithm through a chain of trust [4], [9].

In [10], a composite key management scheme was designed to combine the public key infrastructure (PKI) technique and self-organized certificate chaining technique. In [11], the secure and efficient key management (SEKM) scheme enables share updates among servers in multicast groups.

Our work differs from these related work in the sense that we focus on the use of limited memory space to cache the public keys of different nodes. We further design a public key search technique and a cache update algorithm to achieve optimum caching ratio of public keys from local/remote nodes.

III. Public Key Caching Scheme

We first introduce our notations and variables

- PU_i: public key of node i;
- PR_i: private key of node i;
- $\{pt\}_{PR_i}$: message pt signed by PR_i;
- cm: crypted message;
- $\mathcal{C}_i$: set of nodes whose public keys are cached in node i;
- $\mathcal{N}_i$: set of physical neighbors of node i;
- $\mathcal{R}$: list of nodes that involve in the sequence of KREQ message transmission;
- m: total number of public keys that each node caches;
- ϵ: extensive ($\epsilon = 1$) or simple ($\epsilon = 0$) search;
- γ: ratio of the numbers of keys for local/remote nodes cached by one node;
- TTL: the maximum number of hops a KREQ message may travel.

A. Operational Details of the Proposed Scheme

Assume that every node carries the public keys of some other nodes. Such information can be obtained through pre-deployment key caching or through cache update, which will be discussed later.

KREQ Message Format

Source (S)	Destination (Q)	TTL	List of Routers (R)

KREP Message Format

Source (S)	Destination (Q)	$\{PU_Q\}_{PR_i}$	List of Routers (R)

Fig. 1. Message format of the KREQ and the KREP packets.

We first define two control messages that will be sent between nodes for public keys. These messages are called Key REQuest (KREQ) message and Key REPly (KREP) message. The KREQ message will be transmitted from the node requesting the public key of another node. The format of KREQ message is shown in Fig. 1. In this figure, Source and Destination represent the node requesting the public key and the node's identifier whose public key is being requested; the list of routers ($\mathcal{R}$) represents the nodes who have been passing along the request.

The format of KREP message is similar (see Fig. 1). The KREP message contains the public key of the destination and this key is signed by the current router's private key.

We explain the operational details of the KREQ and the KREP messages with an example. Suppose node S needs to obtain node Q's public key. It sends a KREQ=$\{S, Q, TTL, \mathcal{R} = \Phi\}$ to itself. Every node receiving a KREQ message should check whether it has cached Q's public key. If it does, it will sign the public key of Q with its own private key, store the result onto a KREP message, and return to the previous sender. Otherwise, it will attach its own ID to the list of routers, $\mathcal{R}$, and forwards it to each of the neighbors whose public keys are cached on its memory. The processing of KREP message is simple: a node receiving a KREP message should first authenticate the message, using the public key of the last sender. Then Q's public key will be signed by this node's private key and sent to the previous node according to $\mathcal{R}$.

There is a parameter ϵ, which controls whether a node caching Q's public key should still forward the KREQ message. When $\epsilon = 1$, our scheme operates with extensive search. All nodes will forward the KREQ message. When $\epsilon = 0$, our scheme operates with simple search: only those nodes without Q's public key in their cache will forward the KREQ message.

The details of the above operations are presented in Algorithm 1. Nodes receiving KREP messages should proceed according to Algorithm 2.

B. Cache Update

Each node can update its cache when it receives or overhears a public key for another node through secured exchanges (such as the KREP message). We categorize the public keys that are cached on each node into two groups: local and remote, which

Algorithm 1 Algorithm to Process KREQ Message

Node i receives a KREQ message=$\{S, Q, TTL, \mathcal{R}\}$
$d \leftarrow$ last node ID in $\mathcal{R}$
$\mathcal{R} \leftarrow \{\mathcal{R};\ i\}$
if $Q \in \mathcal{C}_i$ **then**
    Prepare KREP message as $\{S, Q, \{PU_Q\}_{PR_i}, \mathcal{R}\}$
    Node i sends KREP to node d
    if $\epsilon = 0$ **then**
        exit
    end if
end if
$TTL \leftarrow TTL - 1$
if $TTL > 0$ **then**
    Prepare KREQ message as $\{S, Q, TTL, \mathcal{R}\}$
    for each $j \in \mathcal{C}_i \cap \mathcal{N}_i$ **do**
        Node i sends KREQ message to node j
    end for
end if

Algorithm 2 Algorithm to Process KREP Message

Node i receives a KREP message=$\{S, Q, cm, \mathcal{R}\}$
if $i \equiv S$ **then**
    $d \leftarrow$ first node ID in $\mathcal{R}$
    Node i computes $PU_Q = \{cm\}_{PU_d}$, where $d \in \mathcal{C}_i$
else
    $s \leftarrow$ next node ID in $\mathcal{R}$ after i
    $d \leftarrow$ previous node ID in $\mathcal{R}$ ahead of i
    $m \leftarrow \{cm\}_{PU_s}$
    Prepare KREP message as $\{S, Q, \{m\}_{PR_i}, \mathcal{R}\}$
    Node i sends KREP message to node d
end if

represent the public keys of the local nodes that are direct neighbors of the node itself and remote nodes, respectively.

We assume that each node will be preloaded with the public keys of m randomly chosen nodes in the network. Therefore, the ratio of local/remote public keys may not be the specified optimal value γ. As more and more KREQ/KREP exchanges take place, nodes can perform a cache update procedure as follows:

If the current local/remote public key ratio in cache is lower than γ and the public key of a local node is received, the public key will be used to replace one of the remote public keys. Similarly, if the current local/remote public key ratio in cache is higher than γ and the public key of a remote node is received, the public key will be used to replace one of the local public keys.[1] Different methods can be used to choose the key to be replaced, such as last-in-first-out, first-in-first-out, and most rarely used. In this work, we choose a random selection method among the keys in the same category.

This cache update algorithm is described in detail in Algorithm 3. We will investigate the effect of such cache updates in Section IV.

[1]Care needs to be taken to avoid an oscillation effect, in which the key cache is frequently updated with the local/remote ratio fluctuating slightly above and below γ. We leave this to our future work.

Algorithm 3 Algorithm to Update Cache

Node i receives/overhears the public key of node j, PU_j
if $j \notin \mathcal{C}_i$ **then**
    $\gamma_{cur} \leftarrow |\mathcal{C}_i \cup \mathcal{N}_i|/m$
    if $\gamma_{cur} < \gamma$ and $j \in \mathcal{N}_i$ **then**
        Update cache by replacing a randomly selected key that belongs to remote nodes with PU_j
    end if
    if $\gamma_{cur} > \gamma$ and $j \notin \mathcal{N}_i$ **then**
        Update cache by replacing a randomly selected key that belongs to local nodes with PU_j
    end if
end if

IV. Performance Evaluation

Our simulations were performed in Matlab. We believe that the use of other simulators, such as ns2 or OPNET, are unnecessary because we are focusing on high-level public key caching and cache update process in our evaluation. Unless specified otherwise, these are the simulation setups: $N = 200$ nodes are randomly distributed in a network of size 1000 meters by 1000 meters. The radio transmission range is assumed to be 150 meters. Initially each node carries the public keys of a random set (size m) of the nodes in the network. Then we randomly select some source/destination pairs throughout the network. In each of the source/destination pairs, the source needs to request for the public key of the destination.

In our performance evaluation, we focused on finding the number of public key copies (K) for the destination node that are available for the source within a limited number of hops. Another performance metric of our investigation was the optimum local/remote key ratio, γ^*.

A. Available Copies

We investigate the availability of the public keys in this section. In this study, we assume that every node has a γ mix of local/remote nodes' public keys. Therefore, we focus on the stable states of the node caches and we assume that key caches have been assigned according to the local/remote ratio, γ.

In Fig. 2, we compare the public key availability of extensive search ($\epsilon = 1$) and simple search ($\epsilon = 0$) with different TTL values. Naturally, the extensive search returned a larger K because the KREQ message in extensive search can reach more nodes than the simple search. The results in Fig. 2 confirmed such an expectation. Furthermore, as TTL increases, the value of K increases because the source nodes query larger regions.

Another interesting observation from Fig. 2 is the convex shape of the curves: as γ increases from 0 to 1, K increases with γ at the beginning and eventually decreases as γ increases

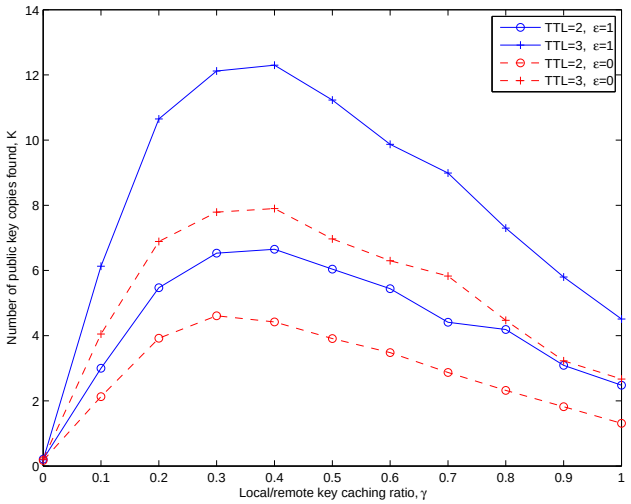

Fig. 2. Comparison of extensive search and simple search when $TTL = 2$ and $TTL = 3$ ($m = 40$).

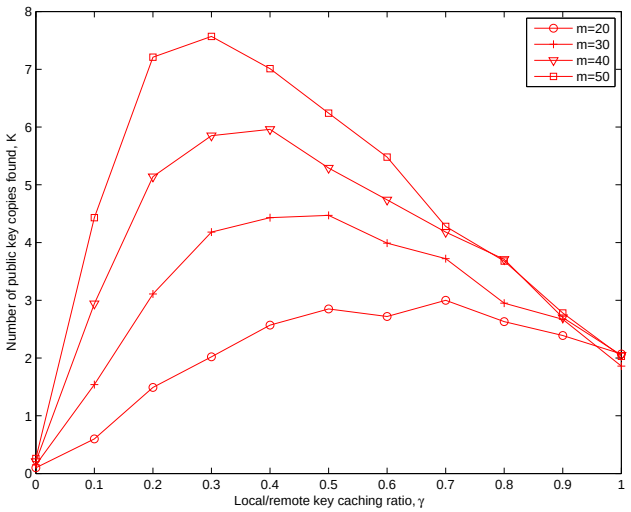

Fig. 3. Comparison of different memory size m ($TTL = 2$, $\epsilon = 1$).

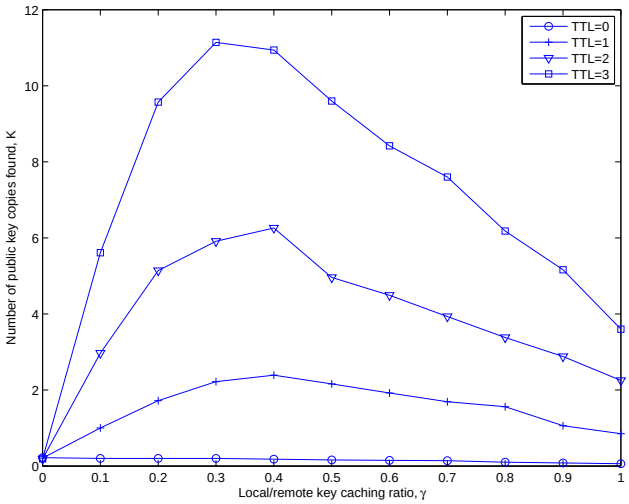

Fig. 4. Comparison of different TTL value TTL ($m = 40$, $\epsilon = 1$).

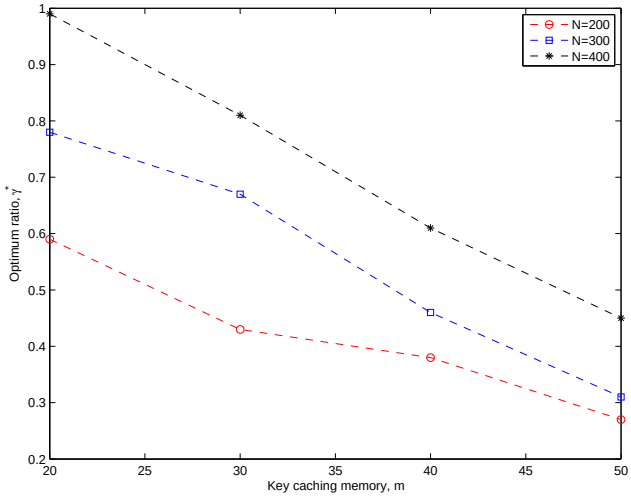

Fig. 5. The optimum ratio γ^* with different N ($TTL = 2$, $\epsilon = 1$).

closer to 1. This can be explained as follows: when γ is close to 0, each node uses its memory to cache mostly the public keys of remote nodes. The secure connectivity in the source node's neighborhood is limited (thus leading to small K), but increasing with γ. On the other hand, when γ is closer to 1, each node uses its memory to cache mostly the public keys of local nodes. The secure connectivity in the source node's neighborhood is good but these nodes are unlikely to have cached the public key of the destination node. Therefore, lowering γ in this region should increase K. Overall, there is an optimum value for γ, which will be investigated in Fig. 5.

Fig. 3 shows the simulation results with different m when $TTL = 2$ and $\epsilon = 1$. As we can see from the figure, K increases with m. This can be explained by the increased number of public keys cached on each node, increasing the chance of finding the public key of the destination. An interesting observation is the K values at $\gamma = 0$ and $\gamma = 1$ for different memory sizes. They remain approximately the same for different m. The reason is that, when every node is caching only the public keys of local (remote) nodes only, the availability of the key largely depends on whether the destination node is a local node.

Fig. 4 presents the simulation results for TTL from 0 to 3. Similar to Fig. 2, K increases with TTL. The results for $TTL = 0$ were presented for comparison purposes only; the source node should at least send the KREQ message to its 1-hop neighborhood.

As we have seen in the above figures, there are always maximum points on the curves. This means that the local/remote ratio should be optimized to maximize the availability of the public keys for the destination node. Such an optimum ratio, γ^*, is investigated in Fig. 5.

In Fig. 5, we present the optimum ratios when we set different numbers for N. For each N, the optimum ratio becomes smaller as the memory size m increases. With the

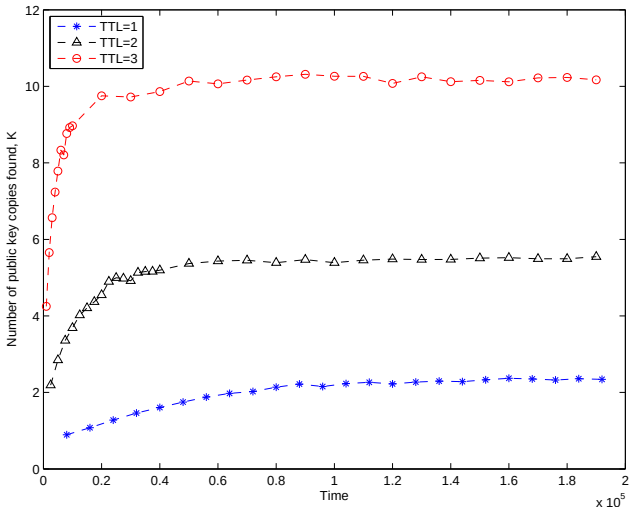

Fig. 6. Cache updates improved the availability of public key copies, K, ($N = 200$, $m = 50$, $\epsilon = 1$).

help from this figure, the optimum local/remote caching ratio can be set for a specific network. For example, for a network with $N = 200$ and $m = 50$, the optimum local/remote caching ratio γ^* should be set to 0.27.

Also can be observed in Fig. 5, the value of γ^* decreases almost linearly with the increase of m. We conjecture that $m\gamma^*$ should be a constant for a network of N nodes. For instance, when $N = 200$, $m\gamma^*$ is about 12-13. The product of $m\gamma^*$ for a network of $N = 400$ is 20-22. These numbers are certainly related to the density of the network, i.e., network region size and wireless transmission range. We leave further investigation as our future work.

B. Dynamic Systems

In this section, we investigate the changes of K as a randomly loaded network of nodes dynamically updating their key caches. According to the simulation result shown in Fig. 5, we chose $\gamma^* = 0.27$ for a network with $N = 200$, $m = 50$, $TTL = 2$, and $\epsilon = 1$.

From the curves in Fig. 6, we can see a clear upward trend of K as time goes on with more and more KREQ/KREP being exchanged. The cache updates improved the availability of required public key copies. After a certain number of cache updates, the value of K stabilizes. The TTL value had a huge effect on the rate of increase for K. As shown in Fig. 6, a larger TTL allows K to reach stable (highest) value faster. This is because, with larger TTL, more nodes have the chance to update their public key caches in each KREQ/KREP message exchange. Thus, the local/remote caching ratios of the nodes in the network approached the designed optimum ratio, γ^*, much faster.

V. Conclusion

Asymmetric cryptography requires the knowledge of the public key from the other party. These public keys are usually certified by a CA in many network systems. In large wireless networks, such CAs may not exist. In this work, we have proposed to use the nodes to serve as the peer CAs. A node requesting the public key of another node may obtain multiple copies from chain of trust. In order to achieve this goal, we have designed a scheme to allow nodes using their limited memory space to cache some of the public keys that they have securely obtained. We have also designed a public key search technique through secure multi-hop paths.

In the investigation of our scheme, we have observed that there exists an optimum caching ratio for the public keys of local nodes and remote nodes. On the one hand, if a node uses all its memory to cache the public keys of local nodes, the hop count of finding enough copies of the public key for a remote node is significantly large. On the other hand, if a node uses all its memory to cache the public keys of remote nodes, a key request may have to travel multiple hops before finding a public key copy. An optimum ratio of the public keys of local nodes and remote nodes should balance these two requirements, allowing the key request to find enough copies within a reasonably small hop count.

In our future work, we will investigate the communication overhead and energy consumption of our scheme and implement it under more realistic network environments and compare its computation and security performance with other state-of-the-art techniques. With the existence of malicious nodes in the network, public key copies may not agree with each other. Trust levels can be maintained and used to select among these copies.

References

[1] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Wormhole attacks in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, pp. 370–380, 2006.

[2] M. Al-Shurman, S.-M. Yoo, and S. Park, "Black hole attack in mobile ad hoc networks," in *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*. New York, NY, USA: ACM, 2004, pp. 96–97.

[3] L. Zhou and Z. J. Haas, "Securing ad hoc networks," *IEEE Network Magazine*, vol. 13, no. 6, November/December 1999.

[4] A. Abdul-Rahman and S. Hailes, "A distributed trust model," in *Proc. of the 1997 Workshop on New Security Paradigms*. ACM, 1997.

[5] S. Yi and R. Kravets, "Moca : Mobile certificate authority for wireless ad hoc networks," in *2nd Annual PKI Research Workshop Program (PKI 03)*, 2003.

[6] M. Omar, Y. Challal, and A. Bouabdallah, "Reliable and fully distributed trust model for mobile ad hoc networks," *Computers and Security*, vol. 28, May/June 2009.

[7] Y. W. Law, R. Corin, S. Etalle, R. Etalle, and P. H. Hartel, "A formally verified decentralized key management architecture for wireless sensor networks," in *Personal Wireless Communications (PWC 2003), Sep 2003. Lecture Notes of Computer Science*. Springer-Verlag, 2003, pp. 27–39.

[8] S. Capkun, L. Buttyan, and J.-P. Hubaux, "Self-organized public-key management for mobile ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 2, pp. 52–64, 2002.

[9] W. Stallings, "The PGP web of trust," *Byte*, vol. 2, pp. 161–162, February 1995.

[10] S. Yi and R. Kravets, "Composite key management for ad hoc networks," in *Proc. of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous04)*, 2004, pp. 52–61.

[11] B. Wu, J. Wu, and E. B. Fern, "Secure and efficient key management in mobile ad hoc networks," in *Proc. of 19th IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2005.