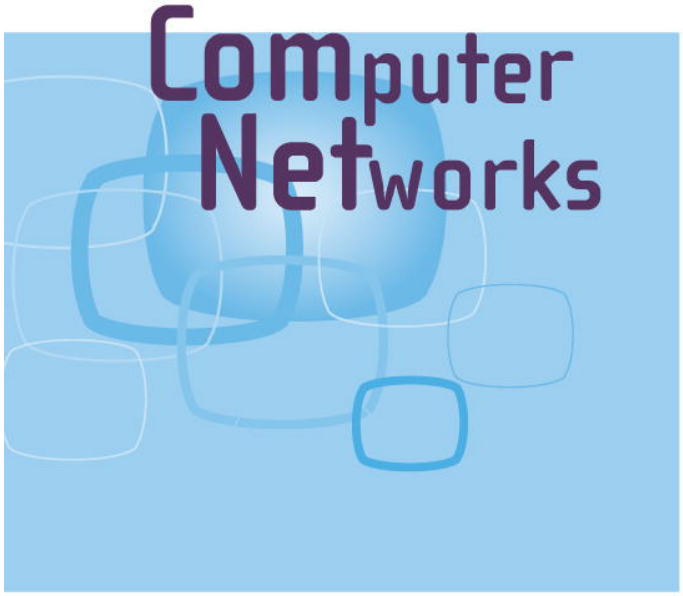
Volume 51 Issue 4 14 March 2007 ISSN 1389-1286



Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 1169–1182
Computer Networks
www.elsevier.com/locate/comnet

Robust self-keying mobile ad hoc networks ☆

Claude Castelluccia [a,1], Nitesh Saxena [b], Jeong Hyun Yi [c,*,1]

[a] *INRIA, France*
[b] *Computer Science Department, University of California, Irvine, United States*
[c] *Networking Technology Lab, Samsung Advanced Institute of Technology, Republic of Korea*

Received 30 April 2006; received in revised form 26 June 2006; accepted 12 July 2006
Available online 15 August 2006

Responsible Editor: X.S. Shen

Abstract

Pairwise key establishment in mobile ad hoc networks allows any pair of nodes to agree upon a shared key. This is an important security service needed to secure routing protocols, and in general to facilitate secure communication among the nodes of the network.

We present two self-keying mechanisms for pairwise key establishment in mobile ad hoc networks which do not require any centralized support. The mechanisms are built using the well-known technique of threshold secret sharing, and are robust and secure against a collusion of up to a certain number of nodes. We evaluate and compare the performance of both the mechanisms in terms of the node admission and pairwise key establishment.

Keywords: Mobile ad hoc networks; Security; Key management

1. Introduction

Mobile ad hoc networks (MANETs) are, by their very nature, vulnerable to many types of attacks. The security of MANETs is often predicated on the availability of efficient key management techniques. However, the usual features of: (1) lack of a centralized authority and (2) dynamic nature of MANETs, represent major obstacles to providing secure, effective and efficient key management. What further complicates the issue is that, in many applications (such as secure routing [9,8,21]) cryptographic keys need to be established *prior* to communication. As a result, standard key exchange solutions, e.g., Station-to-Station protocol [17], are not appropriate since: (1) they require the nodes to interact and (2) they rely on some form of a Public Key Infrastructure (PKI) which is not usually available in MANETs. Related to the latter is the underlying use of public key cryptography which is too expensive for some mobile devices.

☆ A preliminary version of this paper appeared in [6].

* Corresponding author. Tel.: +82 10 2481 0826.
E-mail addresses: claude.castelluccia@inrialpes.fr (C. Castelluccia), nitesh@ics.uci.edu (N. Saxena), jeong.yi@samsung.com (J.H. Yi).

[1] This work has been done while at UC Irvine, United States.

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.07.009

Contributions: This paper proposes two efficient, fully distributed and secure key management mechanisms for MANETs. The so called *self-keying* mechanisms allow nodes in a MANET to establish pairwise keys without communicating and without the need of a PKI. The first mechanism, called *matrix based self-keying* (*MSK*), results from the blending of two well-known techniques: Blom's key pre-distribution [2,13] and threshold secret sharing [25], and the second mechanism, referred to as *polynomial based self-keying* (*PSK*), employs threshold secret sharing using a polynomial. In both *MSK* and *PSK*, a node joins a MANET by receiving a secret token from t different nodes, where t is a security parameter. The schemes are *auto-configurable* in the sense that there is no centralized support required and a node becomes a member only if it is approved by at least t member nodes. Once a node becomes member, it can compute a secret key with any other member without interaction. The proposed schemes are secure against collusion of up to a certain number $(t-1)$ of compromised nodes.

The contribution of this paper is not limited to just the design of secure and efficient key distribution schemes. We also demonstrate our claims of efficiency via extensive analysis and experiments. The schemes have been implemented and tested in a real MANET setting and their performance is compared and analyzed in detail.

Organization: The rest of this paper is organized as follows: Section 2 overviews the related work. Section 3 provides some background on necessary cryptographic building blocks. Sections 5 and 6 present our self-keying mechanisms *MSK* and *PSK* respectively. We discuss some security and other relevant issues of the proposed schemes in Section 7. Finally, in Section 8, we describe the implementation and the performance of our schemes.

2. Related work

Key distribution can be easily achieved if we assume the existence of a PKI. However, this assumption is not realistic in many MANET environments. Zhou and Haas [26] proposed to distribute a Certification Authority (CA) service among several nodes of the network. Although attractive, this idea is not applicable to MANETs. Their approach is hierarchical: only selected nodes can serve as part of the certification authority and thus take part in admission decisions. Moreover, contacting the distributed CA nodes in a MANET setting is difficult since such nodes might be many hops away.

In a related result, Kong et al. [12] developed an interesting Threshold-RSA (TS-RSA) scheme specifically geared for MANETs. Unfortunately, as pointed out in [19,11], TS-RSA is neither verifiable nor secure. An alternative Threshold-DSA (TS-DSA) scheme [19] provides verifiability and, hence, tolerates malicious insiders. However, TS-DSA requires $2t-1$ signers to issue certificates, is heavily interactive and thus become quite inefficient in MANET settings. Moreover, all these solutions require a pair of nodes to perform key exchange protocol to establish shared keys.

Recently, Zhu et al. [27] proposed a pairwise key distribution scheme based on the combination of probabilistic key sharing and threshold secret sharing. However, it is assumed that the nodes are *pre-configured* with some secrets before deployment which is not realistic in a typical MANET environment. Furthermore, two nodes need to communicate over several distinct paths to establish a shared key. In contrast, we do not assume any such pre-configuration and do not require nodes to communicate when establishing a secret key.

Ćapkun et al. proposed a security association establishment protocol that makes use of the mobility of users [5]. Two nodes establish a security association when they are near each other, by using secure channels. As a node moves around, it establishes more and more security associations. When a node needs to establish a secret with another node, there are two possibilities: (1) they already have a security association, or (2) they have no security association and must use the help of "friends" to establish one. Despite the simplicity and elegance of this approach, it is mainly geared for highly mobile MANETs. Furthermore, key derivation among two nodes that do not have a prior security association requires some communication, which is not always practical or even possible.

More closely related results [7,14] present key predistribution schemes based on the schemes by Blom [2] and Blundo et al. [3], respectively. These schemes, unlike the one we propose in this paper, are designed for sensor networks and require a trusted centralized authority for key distribution. Similarly, the trivial solution that consists of configuring each node with pairwise keys, i.e., where a node stores $n-1$ keys, one each it shares with every other node, is not appropriate in MANETs, since (1) this requires a

centralized trusted party to compute and distribute the pairwise keys and (2) every time a new node joins the network, all the current nodes need to be updated. This is clearly not acceptable in a dynamic and volatile environment, like a MANET.

3. Building blocks

This section describes the main techniques used in our proposal, namely threshold secret sharing and Blom key pre-distribution schemes.

Following is the notation used in the rest of this paper:

M_i	member i.e., network node i
t	node admission threshold
λ	number of private keys that M_i must store
N	maximum size of network nodes
NID	network identity
id_i	crypto-based identifier of M_i
K_{ij}	secret key shared between M_i and M_j
$r_i(A)$	row of matrix A for M_i
$ss_i(x)$	secret share of value x for M_i
$pss_j^i(x)$	partial secret share of x for M_i by M_j
SL_i	sponsor list for M_i to reconstruct a secret
$H(x)$	hash value on input x
$E_k(x)$	encryption with a key k on input x
$MAC(k,x)$	message authentication code with key k on input x

3.1. Threshold secret sharing

A (t,n) threshold cryptography allows n parties to share the ability to perform a cryptographic operation in a way that any t parties can perform this operation jointly, whereas no coalition of up to $t-1$ parties can do so. We use Shamir's secret sharing scheme [25] which is based on polynomial interpolation. To distribute shares among n users, a trusted dealer TD chooses a large prime q, and selects a polynomial $f(x) = S + a_1 x + \cdots + a_{t-1}x^{t-1}$ over $\mathbb{Z}_q$ of degree $t-1$ such that $f(0) = S$, where S is the group secret. The TD computes each user's share ss_i such that $ss_i = f(id_i) \pmod q$, and securely transfers ss_i to user M_i. Then, any group of t members who have their shares can recover the secret using the Lagrange interpolation formula: $f(0) = \sum_{i=1}^{t} ss_i\, l_i(0) \pmod q$, where $l_i(x) = \prod_{j=1, j\neq i}^{t} \frac{x - id_j}{id_i - id_j} \pmod q$. To enable the verification of the secret shares, TD publishes a commitment to the polynomial as in *Verifiable Secret Sharing* (VSS) [24]. VSS setup involves a large prime p such that q divides $p-1$ and a generator g which is an element of $\mathbb{Z}_p^*$ of order q. TD computes W_i $(i = 0, \ldots, t-1)$, called the *witness*, such that $W_i = g^{a_i} \pmod p$ and publishes these W_i's in some public domain (e.g., a directory server). On receiving the secret share ss_i from M_i, M_j verifies the correctness of ss_i by checking $g^{ss_i} = \prod_{k=0}^{t-1} (W_k)^{id_i^k} \pmod p$.

3.2. Blom's key pre-distribution

Blom proposed a key pre-distribution scheme that allows any pair of users in a group to compute a pairwise key without communicating [2]. This scheme is secure unless λ users collude (the parameter λ will be defined later). If less than λ users collude, then it is proven that the system is completely secure i.e., the colluding nodes cannot compute any pairwise keys other than their own. However, if λ or more users collude, the whole group is compromised and the colluding users can compute the pairwise keys of all other members.

In Blom's proposal, a trusted dealer TD computes a $\lambda \times N$ matrix B over $\mathbb{Z}_q$, where N is the maximum size of the group, q is a prime, and $q > N$.

One example of such a matrix is a Vandermonde matrix whose element $b_{ij} = (g^j)^i \pmod q$ as seen below, where g is the primitive element of $\mathbb{Z}_q^*$.

$$B = \left[b_{ij} = (g^j)^i \pmod q\right] \text{for } i, j = 1, \ldots, \lambda.$$

Note that this construction requires that $N\lambda < \phi(q)$ i.e., $N\lambda < q - 1$.

Since B is a Vandermonde matrix, it can be shown that any λ columns are linearly independent when $g, g^2, g^3, \ldots, g^N$ are all distinct [15]. The TD then creates a random $\lambda \times \lambda$ symmetric matrix D over $\mathbb{Z}_q$, and computes an $N \times \lambda$ matrix $A = (DB)^{\mathrm{T}}$, where T indicates a transposition of the matrix.

The matrix B is published while the matrix D is kept secret by the TD. Since D is symmetric, the *key matrix* $K = AB$ is also symmetric

$$K = (DB)^{\mathrm{T}}B = B^{\mathrm{T}}D^{\mathrm{T}}B = B^{\mathrm{T}}DB = (AB)^{\mathrm{T}} = K^{\mathrm{T}}.$$

This shows that K is also a symmetric matrix.

We assume that each user, M_i, is defined by an identifier, id_i, such that $0 < i < N$. The TD then sends, over a secret channel, to each user M_i, the

ith row of the matrix A, denoted as $r_i(A)$, i.e., $r_i(A) = [a_{ij}]$ for $j = 1, \ldots, \lambda$.

A user M_i can then compute its key with user M_j as follows: $[K_{ij} = \sum_{\beta=1}^{\lambda} a_{i\beta} \cdot b_{\beta j}]$, where $b_{\beta j}$ is the element of B at row β and column j.

This key can be computed without communication since $b_{\beta j} = (g^j)^{\beta} \pmod q$. Similarly, user M_j can then compute its key with user M_i as follows: $[K_{ji} = \sum_{\beta=1}^{\lambda} a_{j\beta} \cdot b_{\beta i}]$. Since K is symmetric we have $K_{ij} = K_{ji}$, i.e., users M_i and M_j share a secret key. Note that each node does not have to store the whole matrix B only if he knows the public parameter g.

Since each pairwise key is represented by an element in $\mathbb{Z}_q$, q must be selected as the smallest prime number larger than 2^l, where l is the size in bits of the pairwise keys, for example 64.

4. Generic self-keying mechanism

A self-keying mechanism for mobile ad hoc networks consists of various steps. We summarize these steps for a generic mechanism as follows:

1. *Bootstrapping*: The network is bootstrapped by either one single founding member or a set of founding members. The founding member(s) initialize the network by computing the private and corresponding public parameters. The private parameters are secret shared among the founding member(s) in such a way that any set of t members can reconstruct these parameters. The share of the private parameters possessed by each member is referred to as its secret credential.
2. *Member admission*: A prospective member M_{new} who wishes to join the network must be issued its secret credential by the existing member nodes (see Fig. 1). M_{new} initiates the admission protocol by sending a `JOIN_REQ` message to the network. A member node, that receives this `JOIN_REQ` message and approves the admission of M_{new}, replies, over a secure channel (refer to Section 7), with a partial secret credential (derived from its secret credential) for M_{new}. Once M_{new} receives partial secret credentials from at least t different nodes, it uses them to compute it secret credential.
3. *Robustness via verifiability and traceability*: A malicious node can easily launch a denial-of-service (DoS) attack toward a candidate node by inserting incorrect secret shares. This attack would actually deny or disrupt the service to legitimate nodes. To deal with this important problem a node must be able to verify the validity of its reconstructed secret credential before using them. This is what we call *verifiability* in the rest of the paper.Also, when the node detects that its secret credential is not valid, it must be able to trace the bogus shares in order to replace them and/or revoke the malicious participants. This functionality is provided by the *traceability* procedures. Note that verifying the shares' origin, for example via signatures, is not enough to provide traceability since it does not protect against compromised nodes that would signed correctly but send bogus shares. Instead, *traceability* must allow to verify the validity of the shares themselves.Note that *verifiability* is always required. *Traceability* is only necessary when a node detects (from the verifiability service) that its reconstructed secrets are not valid.
4. *Secret key computation*: Each node can use its secret credential and/or the public parameters of the network to compute pairwise keys with other nodes. This allows nodes to securely communicate with other.

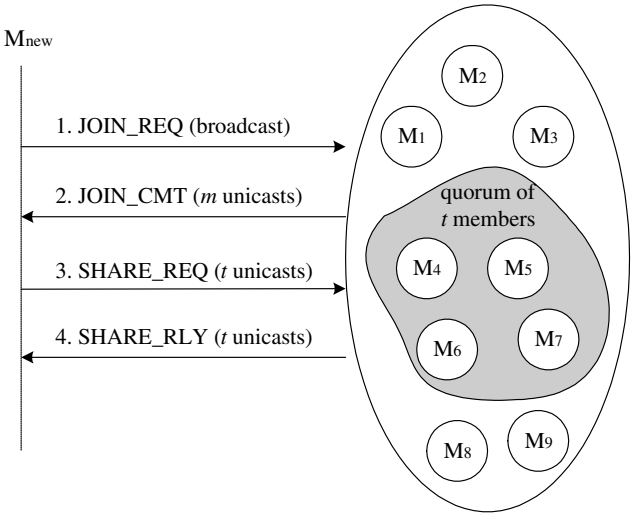


Fig. 1. Abstract Admission Protocol.

5. *MSK*: matrix based self-keying

In this section we describe the *MSK* scheme, which is based on Blom's key pre-distribution described in Section 3.2.

5.1. Bootstrapping

In *MSK* scheme, a network can be bootstrapped (i.e., initialized) by one node (centralized

bootstrapping) or a set of t or more nodes (distributed bootstrapping).

Centralized bootstrapping. The centralized bootstrapping proceeds as follows. First, a founding member FM generates the network parameters, namely N, λ, t, q, p, g, and the matrices $D = [d_{ij}]$ and $B = [b_{ij}]$, where N is the maximum numbers of nodes in the network, (λ, q, p, g) are the security parameters, B is $\lambda \times N$ public matrix such that $b_{ij} = (g^j)^i \pmod q$ for $i,j \in [1, \lambda]$, and D is $\lambda \times \lambda$ symmetric matrix of secrets.

Next, the FM publishes $(N, \lambda, t, q, p, g, B)$ in some public directory, but keeps D secret. It then computes the matrix A such that $A = (DB)^{\mathrm{T}}$ and sends a share of the whole matrix A to each node.

To compute the share $ss_v(D)$ for member M_v, FM selects polynomials for each element d_{ij} of $\lambda \times \lambda$ matrix D. Each polynomial is defined as follows; $f_{d_{ij}}(x) = \sum_{\alpha=0}^{t-1} \delta_{ij}^{(\alpha)} \cdot x^{\alpha} \pmod q$ such that $\delta_{ij}^{(0)} = d_{ij}$. The share of matrix D is made up of shares of its elements $ss_v(d_{ij})$. In other words, $ss_v(D) = [ss_v(d_{ij})] = [f_{d_{ij}}(v)]$ for $i,j = 1, \ldots, \lambda$.

As for $r_v(A)$ such that $r_v(A) = [a_{vj}]$ for $j = 1, \ldots, \lambda$, each element of $r_v(A)$ is simply computed by FM since it knows the secret matrix D. That is, $a_{vj} = \sum_{\beta=1}^{\lambda} d_{j\beta} \cdot b_{\beta v} \pmod q$. Then FM distributes $ss_v(D)$ and $r_v(A)$ to each M_v.

In addition, FM computes VSS witness (which will be used in the traceability procedures defined in Section 5.3.2), $W_{ij}^{(\alpha)}$, as follows: $W_{ij}^{(\alpha)} = g^{\delta_{ij}^{(\alpha)}} \pmod p$ for $i,j \in [1, \lambda]$, $\alpha \in [0, t-1]$.

Distributed bootstrapping. The network can alternatively be bootstrapped by a set of t founding members. The secret matrix D can be generated in fully distributed manner. Note that in the centralized mode, single FM is similar to a trusted third party and is, therefore, a single point of failure. In this proposal, a group of members (the founding members in our scenario) collectively compute shares corresponding to Shamir secret sharing of a random value without a centralized trusted dealer. This procedure is so-called *Joint Secret Sharing* (*JSS*) [22]. The main idea here is that the polynomials for each element d_{ij} of matrix D are constructed among t founding members themselves such that

$$f_{d_{ij}}(x) = f_{d_{ij}[1]}(x) + f_{d_{ij}[2]}(x) + \cdots + f_{d_{ij}[t]}(x),$$

where $f_{d_{ij}[k]}(x)$ is the polynomial of each founding member FM_k over $\mathbb{Z}_q$ for $k = 1, \ldots, t$.

The detailed procedures are as follows. It is assumed that all FM's of the network have previously agreed on the system parameters $(N, \lambda, t, q, p, g, B)$. To compute $ss_v(D)$, each FM_k chooses at random a polynomial $f_{d_{ij}[k]}(x) \in \mathbb{Z}_q$ of degree $(t-1)$ such that $f_{d_{ij}[k]}(x) = \sum_{\alpha=0}^{t-1} \delta_{ij[k]}^{(\alpha)} x^{\alpha} \pmod q$ for $k, v = 1, \ldots, t$, where $\delta_{ij[k]}^{(\alpha)}$ is a random secret that FM_k selects. Then, FM_k computes FM_v's share $\hat{ss}_v^{(k)}(d_{ij}) = f_{d_{ij}[k]}(v)$ for FM_v ($v \in [1, t]$), and *securely* sends it to FM_v (in particular FM_k keeps $\hat{ss}_k^{(k)}$). Note that the share values should be transmitted over the secure channel. Upon receiving $\hat{ss}_v^{(k)}(d_{ij})$, FM_v computes its share $ss_v(d_{ij})$ of the secret d_{ij} as the sum of all shares received: $ss_v(d_{ij}) = \sum_{\alpha=1}^{t} \hat{ss}_v^{(\alpha)}(d_{ij})$.

Next, as for $r_v(A) = [a_{vj}]$ for $j = 1, \ldots, \lambda$, each FM_k computes $a_{vj}^{(k)}$ for FM_v such that $a_{vj}^{(k)} = \sum_{\beta=1}^{\lambda} ss_k(d_{j\beta}) \cdot l_k(0) \cdot b_{\beta v}$ and securely sends it to FM_v. Then, each FM_v gets its own a_{vj} by summing up all a_{vj}^k's, since $a_{vj} = \sum_{k=1}^{t} a_{vj}^k = \sum_{k=1}^{t} \sum_{\beta=1}^{\lambda} ss_k(d_{j\beta}) \cdot l_k(0) \cdot b_{\beta v} = \sum_{\beta=1}^{\lambda} d_{j\beta} \cdot b_{\beta v}$.

Finally, FM_k computes VSS witness $W_{ij[k]}^{(\alpha)}$ of its own polynomial $f_{d_{ij}[k]}(x)$ such that $W_{ij[k]}^{(\alpha)} = g^{\delta_{ij[k]}^{(\alpha)}} \pmod p$ and send it to each FM_v. Then, FM_v obtains the witness $W_{ij}^{(\alpha)}$ of $f_{d_{ij}}(x)$ as follows: $W_{ij}^{(\alpha)} = \prod_{k=1}^{t} W_{ij[k]}^{(\alpha)} \pmod p$. We note that this is actually combined with the procedure for computing $ss_v(D)$ as above.

5.2. Member admission

In order to join the network, a prospective node M_η must collect at least t shares of matrix A's row η from the current member nodes and a valid share of the whole matrix D. Fig. 2 shows the protocol message flow for the member admission process.[2]

1. M_η sends to at least t current member nodes M_v's ($v \in [1, n]$) a signed `JOIN_REQ` message which contains his identity id_η and his public Diffie-Hellman (DH) component y_η ($= g^{x_\eta} \pmod p$). The details about how id_η is generated and verified are discussed in Section 7.
2. After verifying the signed `JOIN_REQ`, the member nodes who wish to participate in the admission process of M_η reply with a signed message containing their respective values id_v and y_v.
3. M_η selects t sponsors $M_\mu (\mu \in {}_R v, |\mu| = t)$, computes a secret key $DHK_{\eta\mu}$ with each of them, forms a sponsor list SL_η which contains the id's of the t

[2] In order to secure the protocol against common *replay* attacks [17], we note that it is necessary to include timestamps, nonces and protocol message identifiers. However, in order to keep our description simple, we omit these values.

$$
\begin{array}{lll}
msg1(M_\eta \rightarrow M_\nu): & REQ = \{id_\eta, y_\eta\},\ S_\eta(REQ) & (1)\\
msg2(M_\eta \leftarrow M_\nu): & REP = \{id_\nu, y_\nu\},\ S_\nu(REP, H(REQ)) & (2)\\
msg3(M_\eta \rightarrow M_\mu): & SL_\eta, MAC(DHK_{\eta\mu},\ H(SL_\eta, msg1, msg2)) & (3)\\
msg4(M_\eta \leftarrow M_\mu): & E_{DHK_{\eta\mu}}\{pss_\mu(r_\eta(D)), ss_\mu(r_\eta(A))\} & (4)
\end{array}
$$

Fig. 2. *MSK* Admission Protocol.

selected sponsors, and replies with an authenticated acknowledgment message to each of them.

4. Each sponsoring node (M_μ) on receiving $msg3$, computes the secret key $DHK_{\eta\mu}$ and replies with row η of it share of the matrix A, $ss_\mu(r_\eta(A))$. The elements of $ss_\mu(r_\eta(A))$ are computed as $ss_\mu(r_\eta(a_{\eta j})) = \sum_{\beta=1}^{\lambda} ss_\mu(d_{j\beta}) \cdot b_{\beta\eta} \pmod q$, for $j = 1, \ldots, \lambda$. This message is encrypted with $DHK_{\eta\mu}$. Each (M_μ) also responds with the *shuffled* partial share of matrix D, $pss_\mu^\eta(D)$, such that $pss_\mu^\eta(D) = [pss_\mu^\eta(d_{ij})] = [ss_\mu(d_{ij}) \cdot l_\mu(id_\eta)] \pmod q$ for $i,j = 1, \ldots, \lambda$. This message is also encrypted using $DHK_{\eta\mu}$. Note that the Lagrange coefficients $l_\mu(id_\eta)$ are publicly known, and therefore, M_η can derive $ss_\mu(d_{ij})$ from $pss_\mu^\eta(d_{ij})$. This can be prevented using the *shuffling* technique proposed in [12] by adding extra random value R_{ij} to each share. These R_{ij}'s are secret values and must sum up to zero by construction. They must be securely shared among the t sponsoring nodes.
5. M_η decrypts the messages it receives from the different nodes and calculates his own $r_\eta(A)$ by adding up all $ss_\mu(r_\eta(A))$'s as follows: $r_\eta(A) = \sum_{\mu=1}^{t} ss_\mu(r_\eta(A)) \cdot l_\mu(0) = [\sum_{\mu=1}^{t} ss_\mu(r_\eta(a_{\eta j})) \cdot l_\mu(0)] \pmod q$ for $j = 1, \ldots, \lambda$. M_η also calculates his own share of the matrix D, $ss_\eta(D)$, by adding up the partial share values such that $ss_\eta(D) = \sum_{\mu=1}^{t} pss_\mu^\eta(D) = [\sum_{\mu=1}^{t} pss_\mu^\eta(d_{ij})] \pmod q$ for $i,j = 1, \ldots, \lambda$.

5.3. Robustness via verifiability and traceability

At the end of the admission protocol, the joining node M_η obtains its $r_\eta(A)$ and $ss_\eta(D)$ from the quorum of t member nodes. Before using $r_\eta(A)$ and $ss_\eta(D)$ for key computation or future admission, the node must verify if they are correctly computed since there might be a malicious responder who participated in this admission process and detect them in the process. We therefore propose following *verifiability* and *traceability* mechanisms.

5.3.1. Verifiability

The VSS technique presented in Section 3.1 will be a useful tool for the verifiability. However, we claim that the verifiability must be a very inexpensive operation since it will be performed frequently (whenever a node joins a network). The proposed mechanism to verify the validity of $r_\eta(A)$ is as follows:

1. *When an existing member node M_μ sends the shares to the node M_η it also sends a well-known message, such as "Welcome to network NID" encrypted with the pairwise key shared between M_μ and M_η (since the M_μ knows the node identifier id_η, it can compute the pairwise key $K_{\mu\eta}$). This will be part of step (4) in Fig. 2.*
2. *After M_η reconstructs its systems secrets $r_\eta(A)$, it can then try to decrypt one of the welcome messages received from the member nodes and verify whether $r_\eta(A)$ is correctly computed.*

Additionally, M_η must verify the validity of the reconstructed secret share $ss_\eta(D)$. If $ss_\eta(D)$ is correct, it can be used for future admission of other nodes. One easy way for M_η to verify the validity of $ss_\eta(D)$ is to try to use it to reconstruct its row of the matrix A (i.e., $r_\eta(A)$) as follows:

> *Let us say that $r_\eta(A)$* was computed from the shares *$ss_a(r_\eta(A))$, $ss_b(r_\eta(A))$, $ss_c(r_\eta(A))$ that it received from M_a, M_b and M_c (for $t = 3$).* M_η *can then compute $r'_\eta(A)$ from the shares $ss_a(r_\eta(A))$,* $ss_b(r_\eta(A))$ and $ss_\eta(r_\eta(A))$ *(that can easily be computed from $ss_\eta(D)$).* If *$r'_\eta(A)$ is equal to $r_\eta(A)$ then the share $ss_\eta(D)$ is correct – otherwise it must be rejected.*

5.3.2. Traceability

The verifiability procedures previously described allow a node to verify the validity of the secret (which is the row of the matrix A and a share of the whole matrix D) that it reconstructed from t shares. However, the above procedure cannot be used to identify the bogus shares, in case the verification procedure fails (i.e., detects that the reconstructed secrets are invalid).

In this section, we present two different traceability procedures. The first – *external attack traceability* – traces external malicious nodes, i.e., malicious

nodes that are not part of the MANET and just try to attack the network by sending bogus shares to new member. The second – *internal attack traceability* – is useful to detect attacks coming from current legitimate member nodes that either turn malicious or get compromised.

Both procedures use the previously described VSS technique in some innovative ways. Since the internal traceability procedure is quite costly, we recommend to use the external attack traceability first and use the internal attack traceability procedure only if the first one turns out to be unsuccessful.

External attack traceability

With this procedure, a node M_η, instead of verifying individual element of a share (row or matrix), verifies the sum of the elements of the share. As a result, instead of applying the VSS technique λ or λ^2 times, we only apply it once. This, of course, improves performance considerably.

More specifically, M_η verifies the validity of the share $ss_\mu(r_\eta(A))$ and $pss^\eta_\mu(D)$ using the VSS technique defined in Section 5.1, as follows:

M_η first computes $\sigma_{ss_\mu(r_\eta(A))}$ by summing up all elements of $ss_\mu(r_\eta(A))$; i.e., $\sigma_{ss_\mu(r_\eta(A))} = \sum_{i=1}^{\lambda} ss_\mu(a_{\eta i}) \pmod q$. Since $W_C^{(\alpha)} = \prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda}(W_{ij}^{(\alpha)})^{b_{j\eta}}$ is *pre-computable*, the validity of $\sigma_{ss_\mu(r_\eta(A))}$ can be verified by checking the following equality:

$$g^{\sigma_{ss_\mu(r_\eta(A))}} \stackrel{?}{=} \prod_{\alpha=0}^{t-1}[W_C^{(\alpha)}]^{id_\mu^\alpha} \pmod p,$$

$$\text{where } W_C^{(\alpha)} = \prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda}(W_{ij}^{(\alpha)})^{b_{j\eta}} \pmod p.$$

Proof. Since $ss_\mu(a_{\eta i}) = \sum_{j=1}^{\lambda} ss_\mu(d_{ij}) \cdot b_{j\eta}$, $ss_\mu(d_{ij}) = f_{d_{ij}}(id_\mu) = \sum_{\alpha=0}^{t-1}\delta_{ij}^{(\alpha)} \cdot id_\mu^\alpha \pmod q$, and $W_C^{(\alpha)} = \prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda}(W_{ij}^{(\alpha)})^{b_{j\eta}}$ for $\alpha = 0,\ldots,t-1$,

$$\begin{aligned}
g^{\sigma_{ss_\mu(r_\eta(A))}} &= \prod_{\alpha=0}^{t-1}[W_C^{(\alpha)}]^{id_\mu^\alpha} \\
&= \prod_{\alpha=0}^{t-1}\left[\prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda}(W_{ij}^{(\alpha)})^{b_{j\eta}}\right]^{id_\mu^\alpha} \\
&= \prod_{\alpha=0}^{t-1}\left[\prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda}(g^{\delta_{ij}^{(\alpha)}})^{b_{j\eta}}\right]^{id_\mu^\alpha} \\
&= g^{\sum_{\alpha=0}^{t-1}\sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda}(\delta_{ij}^{(\alpha)}\cdot b_{j\eta})\cdot id_\mu^\alpha} \\
&= g^{\sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda}\left(\sum_{\alpha=0}^{t-1}\delta_{ij}^{(\alpha)}\cdot id_\mu^\alpha\right)\cdot b_{j\eta}} \\
&= g^{\sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda} ss_\mu(d_{ij})\cdot b_{j\eta}} \\
&= g^{\sum_{i=1}^{\lambda} ss_\mu(a_{\eta i})} \pmod p. \qquad \square
\end{aligned}$$

Similarly, given the precomputed $W_D^{(\alpha)} = \prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda} W_{ij}^{(\alpha)}$, a node can verify the validity of a partial of the matrix D, $pss_\mu^\eta(D)$, after computing $\sigma_{pss_\mu^\eta(D)} = \sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda} pss_\mu^\eta(d_{ij}) \pmod q$, as follows:

$$g^{\sigma_{pss_\mu^\eta(D)}} \stackrel{?}{=} \prod_{\alpha=0}^{t-1}[W_D^{(\alpha)}]^{id_\mu^\alpha \cdot l_\mu(id_\eta)} \pmod p,$$

$$\text{where } W_D^{(\alpha)} = \prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda} W_{ij}^{(\alpha)} \pmod p.$$

Proof. Since $pss_\mu^\eta(d_{ij}) = ss_\mu(d_{ij}) \cdot l_\mu(id_\eta)$, $ss_\mu(d_{ij}) = f_{d_{ij}}(id_\mu) = \sum_{\alpha=0}^{t-1}\delta_{ij}^{(\alpha)} \cdot id_\mu^\alpha \pmod q$, and $W_D^{(\alpha)} = \prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda} W_{ij}^{(\alpha)}$ for $\alpha = 0,\ldots,t-1$,

$$\begin{aligned}
g^{\sigma_{pss_\mu^\eta(D)}} &= \prod_{\alpha=0}^{t-1}[W_D^{(\alpha)}]^{id_\mu^\alpha \cdot l_\mu(id_\eta)} \\
&= \prod_{\alpha=0}^{t-1}\left[\prod_{i=1}^{\lambda}\prod_{j=1}^{\lambda} W_{ij}^{(\alpha)}\right]^{id_\mu^\alpha \cdot l_\mu(id_\eta)} \\
&= \left[g^{\sum_{\alpha=0}^{t-1}\sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda}\delta_{ij}^{(\alpha)}}\right]^{id_\mu^\alpha \cdot l_\mu(id_\eta)} \\
&= g^{\sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda}\left(\sum_{\alpha=0}^{t-1}\delta_{ij}^{(\alpha)}\cdot id_\mu^\alpha\right)\cdot l_\mu(id_\eta)} \\
&= g^{\sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda} ss_\mu(d_{ij}) l_\mu(id_\eta)} \\
&= g^{\sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda} pss_\mu^\eta(d_{ij})} \pmod p. \qquad \square
\end{aligned}$$

If the above verification fails, M_η concludes that M_μ is not a legitimate member node. Otherwise, the malicious node is a group member and thus the following procedure must be used.

Internal attack traceability

If a malicious insider (M_m), who has valid $ss_m(D)$, modifies a value in $ss_m(D)$ so that the sum of $ss_m(D)$ remains unchanged (say, $ss'_m(D)$ such that $\sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda} ss'_m(d_{ij}) = \sum_{i=1}^{\lambda}\sum_{j=1}^{\lambda} ss_m(d_{ij})$), the *external attack traceability* procedure does not work. In order

to protect against such an internal attack, VSS must be applied individually to each of elements of $pss_\mu(r_\eta(A))$ and/or $ss^\eta_\mu(D)$. Since $W_I^{(\alpha)} = \prod_{i=1}^{\lambda} (W_{ij}^{(\alpha)})^{b_{j\eta}}$ is pre-computable, *internal traceability* is provided by checking that

$$g^{ss_\mu(a_{\eta i})} = \prod_{\alpha=0}^{t-1} [W_I^{(\alpha)}]^{id_\mu^\alpha} \quad \text{and}$$

$$g^{pss^\eta_\mu(d_{ij})} = \prod_{\alpha=0}^{t-1} [W_{ij}^{(\alpha)}]^{id_\mu^\alpha l_\mu(id_\eta)} \pmod p$$

$$\text{where } W_I^{(\alpha)} = \prod_{i=1}^{\lambda} (W_{ij}^{(\alpha)})^{b_{j\eta}} \pmod p, \quad i,j \in [1,\lambda].$$

Obviously, these tracing mechanisms for an internal attack are more expensive than the external ones. However, we argue that tracing is an infrequent phenomenon, as an attacker knows that if it performs an internal attack, it will be detected in the process.

5.4. Secret key computation

When a node, M_i, reconstructs its private row of matrix A, $r_i(A) = [a_{i1}, \ldots, a_{i\lambda}]$, he can compute a secret key, K_{ij}, with any other node, M_j, of the network as follows:

Since $a_{ij} = \sum_{\alpha=1}^{\lambda} d_{j\alpha} \cdot b_{\alpha i}$ and $d_{ij} = d_{ji}$,

$$K_{ij} = \sum_{\beta=1}^{\lambda} a_{i\beta} b_{\beta j} = \sum_{\beta=1}^{\lambda} \sum_{\alpha=1}^{\lambda} d_{\beta\alpha} b_{\alpha i} b_{\beta j}$$

$$= \sum_{\alpha=1}^{\lambda} \sum_{\beta=1}^{\lambda} d_{\alpha\beta} b_{\beta j} b_{\alpha i} = \sum_{\alpha=1}^{\lambda} a_{j\alpha} b_{\alpha i} = K_{ji}.$$

Note that these keys do not have to be computed in advance but can be computed *on-the-fly*. The security of this key establishment procedure is unconditional, i.e, it is not based on any security assumption. Refer to [3] for the security arguments.

6. *PSK*: polynomial based self-keying

The various steps of the *PSK* scheme, which is based on polynomial secret sharing, are described in following subsections.

6.1. Bootstrapping

Centralized bootstrapping. The centralized bootstrapping works exactly as described in Section 3.1.

Distributed bootstrapping. A group of t or more founding members employ JSS [22] to collectively compute shares corresponding to Shamir secret sharing of a random value.

6.2. Member admission

In order to join the network, a prospective node M_η must collect at least t partial shares from existing nodes to be able to compute its secret share. Fig. 3 shows the protocol message flow for the member admission process.

1–3. Steps 1–3 are exactly the same as in the *MSK* admission protocol described in Section 5.2.

4. Each sponsoring node (M_μ) on receiving *msg*3, computes the secret key $DHK_{\eta\mu}$ and replies with the *shuffled* partial share [12], $pss_\mu(\eta)$, such that $pss_\mu(\eta) = ss_\mu \cdot l_\mu(id_\eta) \pmod q$. This message is encrypted using $DHK_{\eta\mu}$.

5. M_η decrypts the messages it receives from the different nodes and calculates his own secret share ss_η, by adding up the partial share values such that $ss_\eta = \sum_{\mu=1}^{t} pss_\mu(\eta)$.

6.3. Robustness via verifiability and traceability

M_η can easily validate the acquired secret share by checking if $g^{ss_\eta} = \prod_{k=0}^{t-1} (W_k)^{id_\eta^k} \pmod p$ from the public commitment values. In case this verification fails, M_η can trace the node(s) which sent the fake shares by checking the validity of each of $pss_\mu(\eta)$ values. This can achieved by verifying if $g^{pss_\mu(\eta)} = [\prod_{k=0}^{t-1} (W_k)^{id_\mu^k}]^{l_\mu(id_\eta)} \pmod p$.

6.4. Secret key computation

Any pair of nodes M_i and M_j can establish shared keys using their respective secret shares ss_i, ss_j and the public VSS information as described in Section 3.1. M_i computes $g^{ss_j} = \prod_{k=0}^{t-1} (W_k)^{id_j^k} \pmod p$ from the public commitment values, and exponentiate it to its own share ss_i to get a key $K_{ij} = (g^{ss_j})^{ss_i} \pmod p$. Similarly, M_j computes $g^{ss_i} = \prod_{k=0}^{t-1} (W_k)^{id_i^k} \pmod p$ and exponentiate it to its own share ss_j to get a key $K_{ji} = (g^{ss_i})^{ss_j} \pmod p$. Since, $K_{ij} = K_{ji} = K$, M_i and M_j have a shared secret and they can use $H(K)$ as a symmetric key (where $H()$ is a hash function such as MD5 or SHA-1) to secure their subsequent communication.

Note that the above key establishment mechanism is different from standard Diffie-Hellman key exchange protocol. In the latter, the secret exponents used by the parties are *independently* generated, while in the former, the secret exponents (or the secret shares) are *related* by being points on a polynomial and any set of t of these exponents

$$
\begin{array}{lll}
msg1(M_\eta \rightarrow M_\nu): & REQ = \{id_\eta, y_\eta\},\ S_\eta(REQ) & (1)\\
msg2(M_\eta \leftarrow M_\nu): & REP = \{id_\nu, y_\nu\},\ S_\nu(REP, H(REQ)) & (2)\\
msg3(M_\eta \rightarrow M_\mu): & SL_\eta, MAC(DHK_{\eta\mu},\ H(SL_\eta, msg1, msg2)) & (3)\\
msg4(M_\eta \leftarrow M_\mu): & E_{DHK_{\eta\mu}}\{pss_\mu(\eta)\} & (4)
\end{array}
$$

Fig. 3. *PSK* Admission Protocol.

determine the rest. It is not obvious whether such a usage of related exponents in the key establishment remains secure.

We show, in the theorem to follow, that indeed the proposed key establishment procedure using related exponents remains secure. Basically, we show that our scheme remains secure under the Computational Diffie-Hellman (CDH) assumption[3] in the random oracle model (ROM) [1]. In other words, an adversary who corrupts atmost $t-1$ nodes, cannot distinguish a key K_{IJ} for some uncorrupted user pair (M_I, M_J) from random *even* if he learns all other session keys K_{ij} for $(i,j) \neq (I,J)$, as long as the CDH assumption holds and when hash function is modeled as an ideal random oracle. This is the standard notion for the security of a key establishment protocol and is adopted from [4].

Theorem 1 (security of PSK secret key computation). *Under the CDH Assumption in ROM, there exists no probabilistic polynomial time adversary A, which on inputs of secret keys of t corrupted users, and shared keys K_{ij} between every user pair except K_{IJ} $\{(i,j) \neq (I,J)\}$, is able to distinguish with a non-negligible probability K_{IJ} from a random value.*

Proof. We prove the above claim by contradiction, i.e, we prove that if a polynomial time adversarial algorithm A, which on inputs of secret keys of t corrupted users, and shared keys K_{ij} between every user pair except K_{IJ} $\{(i,j) \neq (I,J)\}$, is able to distinguish with a non-negligible probability K_{IJ} from a random value, then there exists a polynomial time algorithm B, which is able to break the CDH assumption in the random oracle model.

In order to construct the algorithm B which breaks the CDH assumption, we first construct a polynomial time algorithm C, which breaks the Square Computational Diffie-Hellman (SCDH)[4] assumption. The algorithm C runs on input of an SCDH instance $y = g^x \pmod p$, and would translate the adversarial algorithm A into outputting $g^{x^2} \pmod p$.

Without loss of generality, we first assume that the adversary A corrupts $t-1$ players denoted by $M_1, M_2, \ldots, M_{t-1}$. Now, the algorithm C runs as follows:

As in the simulation of Feldman's VSS, C picks $x_1, x_2, \ldots, x_{t-1}$ values corresponding to the secret keys of corrupted users, uniformly at random from $\mathbb{Z}_q$. It then sets $x_i = F(id_i)$, and employs appropriate Lagrange interpolation coefficients in the exponent to compute the public witnesses $g^{A_1}, \ldots, g^{A_{t-1}} \pmod p$, where $F(z) = x + A_1 z + \cdots + A_{t-1} z^{t-1} \pmod q$.

Corresponding to the shared keys K_{ij} between every user pair, C picks a random value R_{ij}, and runs the algorithm A on $x_1, \ldots, x_{t-1}$ and $R_{i,j}$ values. Note that the values $x_1, \ldots, x_{t-1}$ and the witnesses have an identical distribution to an actual run of the Feldman's secret sharing protocol, and therefore A cannot see the difference between C's inputs and actual protocol run. Also, since the K_{ij} values for $(i,j) \neq (I,J)$ are obtained by hashing $g^{x_i x_j}$, the only way A can tell the difference, except with negligible probability, between $K_{i,j}$ and $R_{i,j}$ for $(i,j) \neq (I,J)$, is by querying the random oracle on at least one appropriate $g^{x_i x_j}$ value. If A does tell the difference, then C records $R = g^{x_i x_j}$, and use the following equations to compute g^{x^2},

$$x = \sum_{k=1}^{t-1} x_k l_k^i + x_i l_i^i \pmod q,$$

$$x = \sum_{k=1}^{t-1} x_k l_k^j + x_j l_j^j \pmod q$$

(l_k^i denotes the Lagrange coefficient $l_k^G(0)$, where $G = \{1, \ldots, t-1, i\}$).

Multiplying above two equations, we get

$$x^2 = \left(\sum_{k=1}^{t-1} x_k l_k^i\right)\left(\sum_{k=1}^{t-1} x_k l_k^j\right) + x_i x_j l_i^i l_j^j \pmod q.$$

[3] CDH assumption: In a cyclic group generated by $g \in Z_p^*$ of order q, for $a, b \in \mathbb{Z}_q^*$, given $(g, g^a \pmod p, g^b \pmod p)$, it is hard to compute $g^{ab} \pmod p$.

[4] SCDH assumption: In a cyclic group generated by $g \in Z_p^*$ of order q, for $a \in \mathbb{Z}_q^*$, given $(g, g^a \pmod p)$, it is hard to compute $g^{a^2} \pmod p$.

This implies,

$$g^{x^2} = g^{\left(\sum_{k=1}^{t-1} x_k I_k^i\right)\left(\sum_{k=1}^{t-1} x_k I_k^j\right) R^{I_i^i I_j^j}} \pmod p.$$

If A does not tell the difference between $K_{i,j}$ and $R_{i,j}$ for $(i,j) \neq (I,J)$, then it must tell the difference between $K_{I,J}$ and $R_{I,J}$. However, as above, this is only possible, except with negligible probability, if A queries $g^{x_I x_J}$ to the random oracle. Them C records this value (say K) and computes g^{x^2} similarly as above, using the following equation:

$$g^{x^2} = g^{\left(\sum_{k=1}^{t-1} x_k I_k^I\right)\left(\sum_{k=1}^{t-1} x_k I_k^J\right) K^{I_I^I I_J^J}} \pmod p.$$

Now, we will use C to construct B to break a CDH instance (g^u, g^v). This is very simple as outlined in [16]: B runs C on input g^u, then on g^v, and finally on $g^{u+v} = g^u g^v$, and receives g^{u^2}, g^{v^2}, $g^{(u+v)^2}$, respectively. Now, since $(u+v)^2 = u^2 + v^2 + 2uv \pmod q$, B can easily compute g^{uv} from the outputs of C.

Clearly, $Pr(B) = Pr(C)^3$, where $Pr(B)$, $Pr(C)$, denote the probabilities of success of B and C respectively. □

7. Discussion

7.1. Identifier configuration

In the *MSK* and *PSK* schemes, the identifier id_i of each node M_i must be *unique* and *verifiable*. Otherwise, a malicious node could use the identifier of some other node and get its secret from the member nodes during the admission process.

For unique and unforgeable *id* assignment, we propose to use a solution based on *Crypto-Based ID* (*CBID*) [18]: The id_i is chosen by the node itself from an ephemeral public/private key pair. More specifically, the node computes id_i as follows: $id_i = H_{64}(y_i|NID)$, where y_i is M_i's temporary DH public key in our schemes, *NID* is the network identifier and $H_{64}(\cdot)$ a 64-bit long hash function. When a node contacts the member nodes for admission, it sends its identifier id_i together with its ephemeral public key y_i and signs a challenge sent by the member node. Upon reception of the signature, the member node can verify that the id_i actually belongs to the requesting node (by verifying the signature and that the id_i) was generated as $H_{64}(y_i|NID)$. Note that the y_i does not need to be certified and therefore no PKI is required. The identifier is *verifiable* because a node that does not know the private key, associated with the public key used to generate an *id*, cannot claim to own it. Furthermore since *id* is computed from a hash function, collision probability between two nodes is very low. As a result, the identifier are *statistically unique*. Note that this solution requires that $N = 2^{64}$. However, as we will see this has no effect on the performance or scalability of our proposal.

7.2. Secure channel establishment

In the proposed admission protocols, the channels between the node requesting admission and each of the member nodes must be authenticated and encrypted. It has to be authenticated because each member node must be sure that it is sending the shares to the correct node (i.e., the node that claims to own the identifier). Otherwise, the member node could send the shares to an impersonating node. Similarly, the joining node also needs to authenticate the member nodes. The channel has to be private because otherwise a malicious node that eavesdrops on the shares sent to a node could reconstruct the node's secret and impersonate it.

Establishing an authenticated and private channel usually requires the use of certificates, which bind identities to public keys, and an access to a PKI. However, PKI is not always available in MANET environments. Fortunately in our case, what is really needed is a way to bind an identifier to a public key, where the identifier is a number that identifies one row of the matrix A. This binding is actually provided by *CBID*, described previously. As a result, certificates and PKI are not required. Therefore, the public keys (y_i's) that are sent in message 1 and 2 of the protocols described in Sections 5.2 and 6.2 do not need to be certified.

7.3. Parameters selection

The security of the *MSK* scheme relies on two security parameters t and λ, whereas the *PSK* scheme depends only on t. λ and t denote the number of collusions needed to break these schemes. These parameters should be selected carefully. In particular, it is suggested to set $\lambda = t$. However, more generally, λ should be at least t in the hierarchical MANET settings where only a subset of nodes possesses the ability to admit new nodes. For the evaluation of our schemes (as described in the next section) we set $\lambda \geqslant t$.

8. Performance evaluation

We implemented both *MSK* and *PSK* protocols and evaluated them in a real MANET environment in terms of node admission and pairwise key computation costs.

8.1. Experimental setup

The *MSK* and *PSK* protocol suites are implemented on top of the OpenSSL library [20]. They are written in C for Linux, and consists of about 10,000 lines of code for each. The source code is available at [23].

For the experimental setup, we used a total of five laptops; four laptops with a Pentium-3 800 MHz CPU and 256MB memory and one laptop with a Mobile Pentium 1.8 GHz CPU and 512MB memory. Each device ran Linux 2.4 and was equipped with a 802.11*b* wireless card configured in ad-hoc mode. Specifically, for measuring the admission cost, four laptops with same computing power were used to configure the existing member nodes and the high-end laptop was used for the joining node. In our experiments, each node (except the joining node) was emulated by a daemon and each machine was running up to three daemons. The measurements were performed with the different threshold values t and λ for *MSK*. The size of the parameters q was set to 160 bits and p to 512 or 1024 bits.

8.2. Admission cost

To evaluate the admission cost, we measured the total processing time between the sending of the `JOIN_REQ` by the prospective node and the receiving (plus verification) of acquired credentials (i.e., $r_\eta(A)$ and $ss_\eta(D)$ in *MSK* and ss_η in *PSK*). The resulting measurements include the average computation time of the basic operations, the communication costs such as packet encoding and decoding time, the network delay, and so on.

Fig. 4 shows the average admission time for the joining node for different values of the threshold t. For the *MSK* testing, λ was set to 3, 5, 7 and 9. (In the figure, λ is denoted by L.)

As observed from the graphs, the cost for a node to join the network with *PSK* is cheaper than that of *MSK*. This difference in the costs between *MSK* and *PSK* is even higher for higher threshold values. The reason is quite intuitive: *MSK* requires more computation and bandwidth than *PSK*. More specifically, the *MSK* scheme requires $O(\lambda^2 t)$ *multiplications* and $O(\lambda^2)$ *exponentiations* whereas *PSK* requires only $O(t^2)$ *multiplications* and $O(1)$ *exponentiations*. For the bandwidth costs, refer to Table 1.

This table shows that the *PSK* scheme is very efficient in terms of bandwidth. This is an important property for MANET systems which consist of battery-operated devices, because wireless transmission is considered as the most energy consuming operation.[5]

8.3. Traceability cost

As described in Sections 5.3 and 6.3, the traceability procedures are used to identify the cheating or misbehaving nodes during the admission protocols. This section evaluates the performance of these procedures.

Fig. 5 displays the cost of both the internal and external attack traceability procedures. As for the external attack traceability, denoted as *MSK_EXT*, the cost is slightly expensive than *PSK* since some expensive operations are pre-computable in the latter. In details, the computation complexity for both *MSK_EXT* and *PSK* is the same; i.e., $O(t)$. The cost of the internal traceability procedure with *MSK*, denoted as *MSK_INT*, depends on the value of λ as well as t. As a result, this cost increases when λ gets larger. The complexity of the *MSK* internal attack traceability procedure is $O(\lambda^2 t)$ exponentiations with modulus p. However, we expect these procedures to be executed very infrequently only when the external traceability fails.

8.4. Key computation cost

Table 2 compares the cost of computing a pairwise key in our schemes. The results show that *MSK* performs significantly better than a *PSK* protocol. The achieved gains with $\lambda = 9$ range from 10 ($t = 1$) to 13 ($t = 9$), and from 305 to 307 for 512-bit and 1024-bit p, respectively. In other words, *MSK* is 10–307 times faster than *PSK* when establishing a shared secret key.

These results were actually expected because in *MSK* the pairwise computation requires only $O(\lambda)$ modular *multiplications* where the modulus size is 160 bits. In contrast, *PSK* requires $O(t)$ expensive

[5] It has been shown that sending one bit of data is roughly equivalent to adding 1000 32-bit numbers [19].

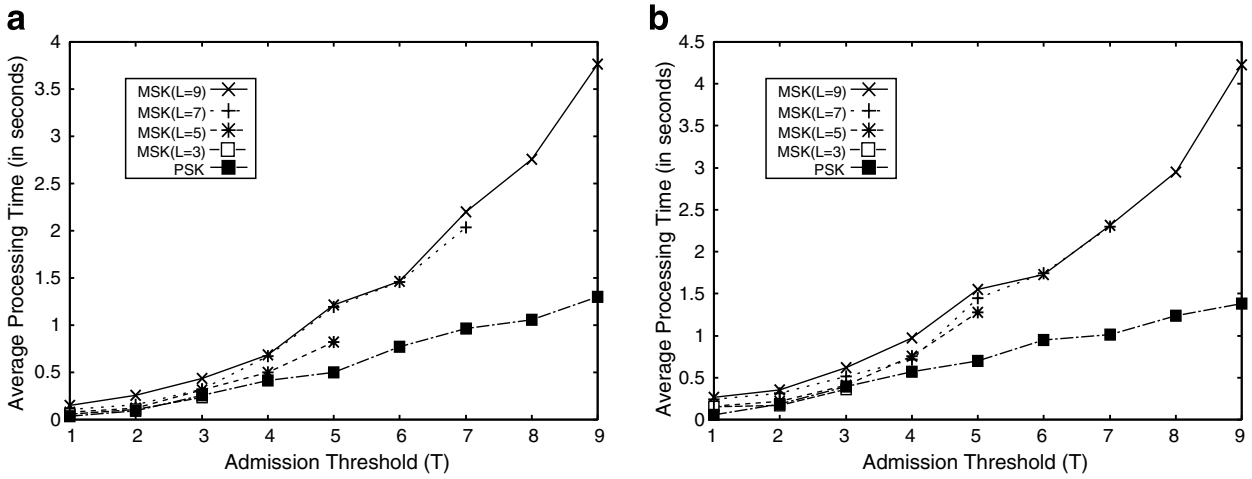


Fig. 4. Admission cost (a): $|p| = 512$ and (b) $|p| = 1024$.

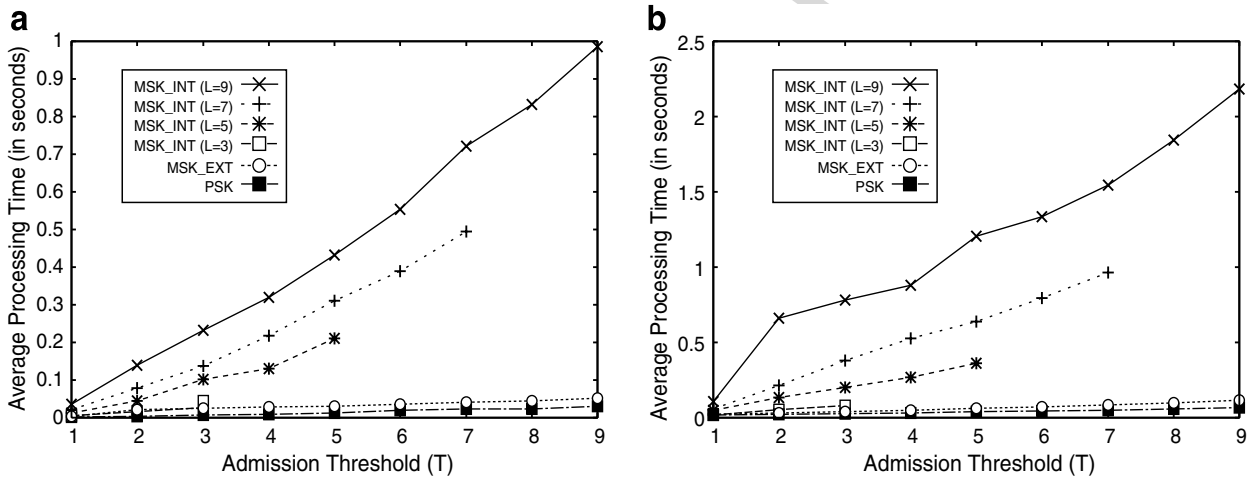


Fig. 5. Traceability cost: (a) $|p| = 512$ and (b) $|p| = 1024$.

Table 1
Bandwidth comparison

	MSK	*PSK*
Admission	$O(\lambda t\|q\|) + O(\lambda^2\|q\|)$	$O(t\|q\|)$
Shuffling	$O(\lambda^2 t^2\|q\|)$	$O(t^2\|q\|)$

Table 2
Key computation cost (in ms, P4-3.0 GHz, 1 GB memory)

t	*MSK* ($\lambda \geqslant t$)				*PSK*	
	$\lambda = 3$	$\lambda = 5$	$\lambda = 7$	$\lambda = 9$	$\|p\| = 512$	$\|p\| = 1024$
1	0.0371	0.0301	0.0430	0.0550	0.574	17.780
2	0.0398	0.0415	0.0506	0.0570	0.683	18.150
3	0.0436	0.0424	0.0568	0.0564	0.713	18.180
4	–	0.0365	0.0595	0.0655	0.663	18.220
5	–	0.0431	0.0565	0.0629	0.753	18.370
6	–	–	0.0628	0.0563	0.772	18.450
7	–	–	0.0562	0.0629	0.782	18.570
8	–	–	–	0.0644	0.851	18.540
9	–	–	–	0.0637	0.871	19.120

modular *exponentiations* with a modulus size of 512 or 1024 bits.

9. Conclusion

We presented distributed solutions to the key predistribution problem in MANETs. Our self-keying solutions, *MSK* and *PSK*, are based on the secret sharing techniques and are secure against collusive attacks by a certain threshold of nodes. The solutions allow any pair of nodes in the network to establish shared keys *without communication*, as opposed to the standard Diffie-Hellman key exchange protocols. We implemented the *MSK* and *PSK* schemes and evaluated them in real MANET setting. Our analysis show that *MSK* fares better than *PSK* as far as the pairwise key establishment costs are concerned. However, in terms of the node admission costs, the latter outperforms the former. Based on

this analysis, we conclude that the *MSK* scheme is well-suited for MANET applications where node admission is not a frequent operation, whereas the *PSK* scheme is more applicable for highly dynamic MANETs consisting of mobile devices with reasonably high computation power.

References

[1] M. Bellare, P. Rogaway, Random oracles are practical: a paradigm for designing efficient protocols, in: ACM Conference on Computer and Communications Security, 1993, pp. 62–73.

[2] R. Blom, An optimal class of symmetric key generation systems, in: EUROCRYPT'84, LNCS, IACR, 1984.

[3] C. Blundo, A.D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, perfectly-secure key distribution for dynamic conferences, in: Advances in Cryptology – CRYPTO'92, 1992, pp. 471–486.

[4] R. Canetti, H. Krawczyk, Analysis of key-exchange protocols and their use for building secure channels, in: EUROCRYPT'01, Springer-Verlag, 2001, pp. 453–474.

[5] S. Capkun, J.-P. Hubaux, L. Buttyan, Mobility helps security in ad hoc networks, in: 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03), 2003, pp. 46–56.

[6] C. Castelluccia, N. Saxena, J.H. Yi, Self-configurable key pre-distribution in mobile ad hoc networks, in: IFIP Networking Conference, May 2005, pp. 1083–1095.

[7] W. Du, J. Deng, Y.S. Han, P.K. Varshney, A pairwise key pre-distribution scheme for wireless sensor networks, in: Jajodia et al. [10], pp. 42–51.

[8] Y.-C. Hu, D.B. Johnson, A. Perrig, Sead: secure efficient distance vector routing in mobile wireless ad hoc networks, in: Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02), June 2002, pp. 3–13.

[9] Y.-C. Hu, A. Perrig, D.B. Johnson, Ariadne: a secure on-demand routing protocol for ad hoc networks, in: Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking, MobiCom 2002.

[10] S. Jajodia, V. Atluri, T. Jaeger (Eds.), Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, 27–30 October 2003, ACM, 2003.

[11] S. Jarecki, N. Saxena, J.H. Yi, An attack on the proactive RSA signature scheme in the URSA ad hoc network access control protocol, in: ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN), October 2004, pp. 1–9.

[12] J. Kong, P. Zerfos, H. Luo, S. Lu, L. Zhang, Providing robust and ubiquitous security support for MANET, in: IEEE 9th International Conference on Network Protocols (ICNP), 2001.

[13] T. Leighton, S. Micali, Secret-key agreement without public-key cryptography, in: CRYPTO'93, 1993.

[14] D. Liu, P. Ning, Establishing pairwise keys in distributed sensor networks, in Jajodia et al. [10], pp. 52–61.

[15] F.J. MacWilliams, N. Sloane, The Theory of Error-Correcting Codes, North Holland, Amsterdam, 1997.

[16] U.M. Maurer, S. Wolf, Diffie-Hellman oracles, in: CRYPTO'96, LNCS, vol. 1109, 1996, pp. 268–282.

[17] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, CRC Press Series on Discrete Mathematics and Its Applications, 1997, ISBN 0-8493-8523-7.

[18] G. Montenegro, C. Castelluccia, Crypto-based identifiers (cbids): concepts and applications, ACM TISSEC 7 (1) (2004).

[19] M. Narasimha, G. Tsudik, J.H. Yi, On the utility of distributed cryptography in P2P and MANETs: the case of membership control, in: IEEE International Conference on Network Protocol (ICNP), November 2003, pp. 336–345.

[20] OpenSSL Project, http://www.openssl.org/.

[21] P. Papadimitratos, Z. Haas, Secure Routing for Mobile Ad Hoc Networks, 2002.

[22] T.P. Pedersen, A threshold cryptosystem without a trusted party, in: D. Davies (Ed.), EUROCRYPT'91, IACR, 1991, LNCS, vol. 547, pp. 552–526.

[23] Peer Group Admission Control Project. Available from: <http://sconce.ics.uci.edu/gac>.

[24] P. Feldman, A practical scheme for non-interactive verifiable secret sharing, in: 28th Symposium on Foundations of Computer Science (FOCS), 1987, pp. 427–437.

[25] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.

[26] L. Zhou, Z.J. Haas, Securing ad hoc networks, IEEE Network Mag. 13 (6) (1999) 24–30.

[27] S. Zhu, S. Xu, S. Setia, S. Jajodia, Establishing pair-wise keys for secure communication in ad hoc networks: a probabilistic approach, in: IEEE International Conference on Network Protocol (ICNP), November 2003.

Claude Castelluccia is a senior research scientist at INRIA, France. He received an engineering degree from the Universite de Technologie de Compiegne (90), France, a Master of Science in EE from Florida Atlantic University (92) and a Ph.D. in Computer Science from INRIA (96). He was a post-doctoral fellow at Stanford University in 1997 and a visiting researcher at University of California Irvine from 2003 to 2005. His research interests are in network security, applied cryptography, mobile/wireless networking. wireless sensor networks and RFID.

Nitesh Saxena is an assistant professor in the department of Computer and Information Science at Polytechnic University, starting Fall 2006. He obtained his Ph.D. in Information and Computer Science from University of California, Irvine, in summer 2006. He holds an M.S. degree in Computer Science from UC Santa Barbara, and a Bachelor's degree in Mathematics and Computing from Indian Institute of Technology, Kharagpur, India. Nitesh's research spans all areas of information security with core emphasis on network and distributed system security and applied cryptography. Nitesh's Ph.D. dissertation entitled "Decentralized Security Services" has been nominated for the ACM Dissertation Award for the year 2006.

Jeong Hyun Yi is a principal researcher in Samsung Advanced Institute of Technology (SAIT). He received his Ph.D. in Information and Computer Science with his advisor, Dr. Gene Tsudik, from University of California, Irvine in 2005. He received his M.S. and B.S. in Computer Science at Soongsil University, Korea in 1995 and 1993, respectively. He was a senior researcher at Electronics and Telecommunication Research Institute (ETRI), Korea from 1995 to 2001 and a guest researcher at National Institute of Standards and Technology (NIST), MD, USA from 2000 to 2001. His research interests are in network security, applied cryptography, ubiquitous computing, RFID and wireless sensor networks.