Self-Organized Public-Key Management for Mobile Ad Hoc Networks

Srdjan Capkun, *Student Member*, *IEEE*, Levente Buttyán, *Student Member*, *IEEE*, and Jean-Pierre Hubaux, *Senior Member*, *IEEE*

Abstract—In contrast with conventional networks, mobile ad hoc networks usually do not provide online access to trusted authorities or to centralized servers, and they exhibit frequent partitioning due to link and node failures and to node mobility. For these reasons, traditional security solutions that require online trusted authorities or certificate repositories are not well-suited for securing ad hoc networks. In this paper, we propose a fully self-organized public-key management system that allows users to generate their public-private key pairs, to issue certificates, and to perform authentication regardless of the network partitions and without any centralized services. Furthermore, our approach does not require any trusted authority, not even in the system initialization phase.

Index Terms—Mobile ad hoc networks, self-organization, security, key authentication, public-key cryptography, PGP.

1 Introduction

By definition, a mobile ad hoc network [1], [2], [3] does not rely on any fixed infrastructure; instead, all networking functions (e.g., routing, mobility management, etc.) are performed by the nodes themselves in a self-organizing manner. For this reason, securing mobile ad hoc networks is challenging and, as we show in this paper, in some applications this requires a shift in paradigms with respect to the traditional security solutions for wireline networks. Meanwhile, we still rely on traditional cryptographic primitives.

In our view, there are two extreme ways to introduce security in mobile ad hoc networks: 1) through a single authority domain, where certificates and/or keys are issued by a single authority, typically, in the system setup phase or 2) through full self-organization, where security does not rely on any trusted authority or fixed server, not even in the initialization phase.

In this paper, we take the second approach and we propose a *self-organizing public-key management system* that allows users to create, store, distribute, and revoke their public keys without the help of any trusted authority or fixed server. Moreover, in our solution, we do not assign specific missions to a subset of nodes (i.e., all the nodes have the same role). Our main motivation for taking this approach comes from the self-organized nature of mobile ad hoc networks and from the need to allow users to fully control the security settings of the system. As such, our approach is developed mainly for "open" networks, in which users can join and leave the network without any centralized control.

The main problem of any public-key based security system is to make each user's public key available to others in such a way that its authenticity is verifiable. In mobile ad hoc networks, this problem becomes even more difficult to solve because of the absence of centralized services and possible network partitions. More precisely, two users willing to authenticate each other are likely to have access only to a subset of nodes of the network (possibly those in their geographic neighborhood). The best known approach to the public-key management problem is based on public-key certificates [4]. A public-key certificate is a data structure in which a public key is bound to an identity (and possibly to some other attributes) by the digital signature of the issuer of the certificate.

In our system, like in PGP [5], users' public and private keys are created by the users themselves. For simplicity, we assume that each honest user owns a single mobile node. Hence, we will use the same identifier for the user and her node (i.e., both user u and her node will be denoted by u). Unlike in PGP, where certificates are mainly stored in centralized certificate repositories, certificates in our system are stored and distributed by the nodes in a fully self-organized manner. Each certificate is issued with a limited validity period and therefore contains its issuing and expiration times. Before a certificate expires, its issuer issues an updated version of the same certificate, which contains an extended expiration time. We call this updated version the *certificate update*. Each node periodically issues certificate updates, as long as its owner considers that the user-key bindings contained in these certificates are correct.

In our system, key authentication is performed via chains of public-key certificates in the following way: When a user u wants to obtain the public key of another user v, she acquires a chain of valid public-key certificates such that:

- *S. Capkun and J.-P. Hubaux are with the Laboratory for Computer Communications and Applications (LCA), School of Information and Communication Sciences, Swiss Federal Institute of Technology—Lausanne (EPFL), CH-1015 Lausanne, Switzerland. E-mail: {srdan.capkun, jean-pierre.hubaux}@epfl.ch.*
- *L. Buttyán is with the Budapest University of Technology and Economics, Department of Telecommunications, Magyar tudosok krt. 2, H-1117 Budapest, Hungary. E-mail: buttyan@hit.bme.hu.*

Manuscript received 19 June 2002; revised 11 Dec. 2002; accepted 24 Jan. 2003.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number 5-062002.

 Published by the IEEE CS, CASS, ComSoc, IES, & SPS

1. The first certificate of the chain can be directly verified by u, by using a public key that u holds and trusts (e.g., her own public key).
2. Each remaining certificate can be verified using the public key contained in the previous certificate of the chain.
3. The last certificate contains the public key of the target user v.

To correctly perform authentication via a certificate chain, a node needs to check that: 1) all the certificates on the chain are *valid* (i.e., have not been revoked) and 2) all the certificates on the chain are *correct* (i.e., not false; the certificates contain correct user-key bindings). To find appropriate certificate chains to other users, each node maintains two local certificate repositories: the *nonupdated certificate repository* and the *updated certificate repository*. The nonupdated certificate repository of a node contains expired certificates that the node does not keep updated. The reason for collecting and not updating expired certificates is that most of the certificates will permanently be renewed by their issuers and only a few will be revoked. Therefore, the nonupdated repositories provide the nodes with a very good estimate of the certificate graph. As we show later in Section 3, this information helps nodes to perform authentication. The updated certificate repository of a node contains a subset of certificates that the node keeps updated. This means that the node requests the updates for the certificates contained in its updated repository from their issuers, when or before they expire. The selection of certificates into the node's updated repository is performed according to an appropriate algorithm.

When a user u wants to authenticate a public key K_v of another user v, both nodes merge their updated certificate repositories and u tries to find a certificate chain to v in the merged repository. If found, this chain contains only updated certificates because it is constructed in the updated repositories. To authenticate K_v, u then further checks whether the certificates on the chain have been revoked (since the last update) and the user-key bindings in the certificates are correct. As we describe in Sections 3.4 and 3.5, u performs both validity and correctness checks locally. In Section 4.5, we present an algorithm for the construction of users' updated repositories that we call Maximum Degree. As we show through simulations, with this algorithm, there is a high probability of finding certificate chains between the users in their merged updated repositories even if the size of the users' updated repositories is small.

If the authentication of K_v through the updated certificate repositories fails, node u tries to find certificate chains to v in its (us) joint updated and nonupdated repositories. If u finds a chain to v, this chain will likely contain some expired certificates because it is constructed in the updated and nonupdated repositories. To complete the authentication, u requests, from their issuers, the updates of the expired certificates that lay on the chain and checks their correctness. If the certificates are both valid and correct, u authenticates K_v. Here again, u performs the certificate correctness check locally. If node u cannot find any certificate chain to K_v, it aborts the authentication.

In our system, certificate revocation is an important mechanism. We enable two types of certificate revocation: *explicit* and *implicit*. The issuer explicitly revokes a certificate by issuing a revocation statement and by sending it to the nodes who stored the certificate in question. The implicit revocation relies on the expiration time contained in the certificates. Every certificate whose expiration time passes is implicitly revoked; this second mechanism is straightforward, but requires some loose time synchronization of the nodes.

The paper is organized as follows: In Section 2, we present an overview of the existing proposals for public-key management in mobile ad hoc networks. In Section 3, we describe the basic self-organized public-key management scheme. In Section 4, we present simulation results of the performance of our system. In Section 5, we present an analysis of some properties of the local repository construction algorithms. In Section 6, we conclude and we give some remarks on our future work.

2 State of the Art

Solutions to the problem of public-key management in mobile ad hoc networks have already been proposed; they are briefly summarized in this section.

In [6], the authors propose a distributed public-key management service for ad hoc networks. The service, as a whole, has a public/private key pair K/k that is used to verify/sign public-key certificates of the network nodes. It is assumed that all nodes in the system know the public-key K and trust any certificates signed using the corresponding private key k. The private key k is divided into n shares using an $(n, t+1)$ *threshold cryptography* scheme, and the shares are assigned to n arbitrarily chosen nodes, called servers. For the service to sign a certificate, each server generates a partial signature for the certificate using its private key share and submits the partial signature to a combiner that computes the signature from the partial signatures. The application of threshold cryptography ensures that the system can tolerate a certain number $t < n$ of compromised servers in the sense that at least $t+1$ partial signatures are needed to compute a correct signature. Besides threshold signatures, the proposed key management service also employs *proactive share refreshing* in order to tolerate *mobile adversaries* and to adapt its configuration to changes in the network. This proposal, however, assumes that there is an authority that initially empowers the servers and that some of the nodes must behave as servers.

A more recent proposal [7] describes a similar approach, but it provides a more fair distribution of the burden by allowing *any* node to carry a share of the private key of the service. The advantage is increased availability since now *any* $t+1$ nodes in the local neighborhood of the requesting node can issue or renew a certificate. Another novelty is that any node not yet possessing a share can obtain a share from any group of at least $t+1$ nodes already possessing a share. However, just like in [6], the first $t+1$ nodes must be initialized by a trusted authority. In addition to this drawback, there are two problems with this proposal: First,

the number t must be a trade off between availability and robustness; but, it is not clear how the value of t can be changed when the overall number of nodes significantly increases (or decreases). Second, the system seems to be vulnerable to the Sybil attack [8]: An attacker can take as many identities as necessary to collect enough shares and reconstruct the system's private key.

Two other approaches, both originally designed for the address ownership problem in Mobile IPv6, are described in [9] and [10]. The main idea behind these approaches is to avoid certificates altogether and bind the name (IP address) of a node to its public key by deriving the former from the latter in a cryptographically verifiable way: First, the public key is hashed with a cryptographic hash function, and then (part of) the hash value is used as part of the IP address of the node. This approach makes sense at the network layer, where names (IP addresses) are handled by machines. But, it does not seem to be applicable at the application layer, where names often refer to and are processed by people, and cannot thus be computed by hashing a public key. In addition, if the public key of a node is compromised, then its revocation requires the node to change its name, which could be impractical in some applications.

The work described in this paper is part of the Terminodes project [11], [12]. Several results related to security have already been published, for key management [13], [14] and for cooperation among the nodes [15].

More work in the area of ad hoc network security and mobile device security has been reported in [16], [17], [18], [19], [20], [21], [22], [23], [24], but these papers are only loosely related to the security problems addressed in this paper.

3 Basic Operations of Our Solution

In what follows, it will be useful to think of our scheme in terms of an abstract model. In this model, the public keys and the certificates of the system are represented as a directed graph $G(V, E)$, where V and E stand for the set of vertices and the set of edges, respectively.[1] We call this graph the *certificate graph*. The vertices of the certificate graph represent public keys and the edges represent certificates. More precisely, there is a directed edge from vertex K_u to vertex K_w if there is a certficate signed with the private key of u that binds K_w to an identity. A certificate chain from a public key K_u to another public-key K_v is represented by a directed path from vertex K_u to vertex K_v in G. Thus, the existence of a certificate chain from K_u to K_v means that vertex K_v is reachable from vertex K_u in G (denoted in the sequel $K_u \rightarrow_G K_v$). In the rest of the paper, the certificate graph G designates the graph comprising only the valid (not expired) certificates of the whole network. In our model, we represent the updated and the nonupdated certificate repositories of user u by the certificate graphs G_u and G_u^N, respectively. Therefore, for any u, G_u is a subgraph of G, but G_u^N is not necessarily a subgraph of G, as it may also contain some implicitly revoked certificates.

Now, we briefly describe the unfolding of the basic operations of our scheme; they will be described in more detail in the sections that follow. As shown in Fig. 1, the initial phase of our scheme is executed in four steps: the creation of public/private key pairs, the issuing of certificates, the certificate exchange, and the creation of nodes' updated certificate repositories. In Step 0, the user creates her own public/private key pair. In Step 1, she issues public-key certificates based on her knowledge about other users' public keys. Note that the issuing of public-key certificates also continues when the system is fully operational (i.e., when the updated and nonupdated repositories are already constructed) as users get more information about other users' public keys. During this process, the certificate graph G is created. The speed of the creation of a usable (i.e., sufficiently connected) certificate graph heavily depends on the motivation of the users to issue certificates. In Step 2, the node performs the certificate exchange. During this step, the node collects certificates and thus creates its nonupdated certificate repository. Along with the creation of new certificates, the certificate exchange also continues even when the system is fully operational. This means that nodes' nonupdated repositories will be continuously upgraded with new certificates. As we show later through simulations, nodes' mobility determines the speed at which certificates are accumulated by the nodes themselves. In Step 3, the node constructs its updated certificate repository. The node can perform this operation in two ways, either by communicating with its certificate graph neighbors (Step 3a), or by applying the repository construction algorithm (described in Section 4.5) on the nonupdated certificate repository (Step 3b). When the node constructed its updated certificate repository, it is ready to perform authentication.

In our solution, the authentication is performed in the following way: As already mentioned in the introduction, when a user u wants to verify the authenticity of the public-key K_v of another user v, u tries to find a directed path from K_u to K_v in $G_u \cup G_v$. The certificates on this path are then used by u to authenticate K_v. An example of a certificate graph with updated local repositories of the users is shown on Fig. 2.

If there is no path from K_u to K_v in $G_u \cup G_v$, u tries to find a path from K_u to K_v in $G_u \cup G_u^N$. If such a path is found, u updates the expired certificates, checks their correctness and performs authentication. If there is no path from K_u to K_v in $G_u \cup G_u^N$, u fails to authenticate K_v. We will now provide a detailed description of each operation.

3.1 Creation of Public Keys and Public-Key Certificates

The public key and the corresponding private key of each user is created locally by the user herself. Public-key certificates are issued by the users. If a user u believes that a given public key K_v belongs to a given user v, then u can issue a public-key certificate in which K_v is bound to v by the signature of u. Certificates are issued with a limited validity period T_V, and each certificate contains its issuing and expiration times. Here, for simplicity, we assume that

1. For simplicity, we assume that each user generates a single (public, private) key pair and, therefore, is represented by a single vertex on the graph.

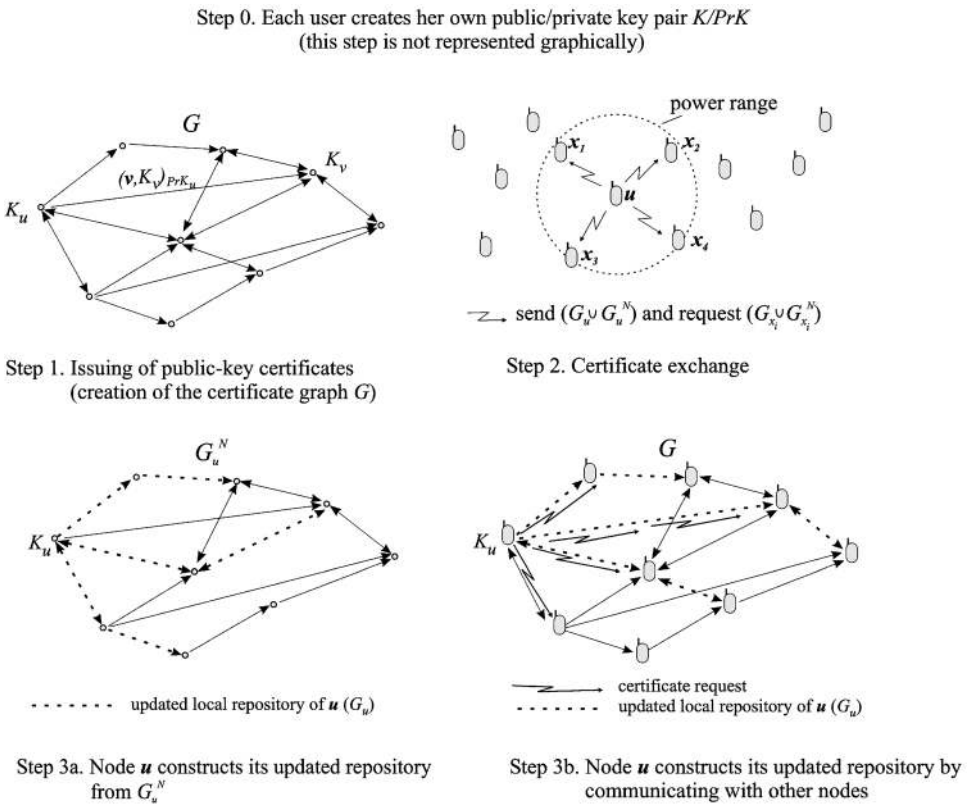

Fig. 1. The creation of nodes' updated and nonupdated repositories.

all certificates are issued with the same validity period. When a certificate expires and its issuer believes that the user-key binding certified by that certificate is still valid, the node issues a new updated version of the same certificate. The updated certificate contains the same user-key binding as the old certificate, but has a new issuing time and an expiration time that is extended by T_V.

There may be several reasons for u to believe that K_v belongs to v. For example, u and v may have exchanged their keys through a side channel (e.g., over an infrared channel at the time of a physical encounter). The dynamic nature of ad hoc networks enables users to gather more experience about other users, issue a higher number of certificates, and better evaluate their confidence in the certificates they issue.

The characteristics of the certificate graphs created in mobile ad hoc networks will depend on the applications for which the networks are used. A useful example of the possible characteristics of the certificate graphs is the PGP certificate graph, as this graph is the only known example of a certificate graph created in a self-organized manner. In our previous work [14], we analyzed the characteristics of the PGP certificate graph. In that work, we showed that the PGP certificate graph exhibits the small-world phenomenon [25], [26], [27] (i.e., this graph has 1) a small average shortest path length that scales logarithmically with its size and 2) clustered vertices). The results of the analysis of the PGP certificate graph are in line with our expectations. Clearly, the PGP certificate graph reflects the existing social relationships between the users. In a way, these certificates can be seen as an indication of the users' existing personal relationships. We argue that the small-world phenomenon naturally emerges in self-organized security systems such as ad hoc networks, as well because of social relationships. However, certificate graphs in mobile ad hoc networks will be different from the PGP certificate graph in a sense that they will be enriched by a number of certificates that will be issued as a result of the mobility of the nodes. More precisely, mobile nodes will enable users to exchange their public keys whenever they meet, provided that they can establish communication over some short range (possibly secure) channel. This will lead to the creation of a small-world graph which is better connected than the PGP certificate graph. In [14], we also presented a model for the creation of PGP-like small-world graphs. This model allows for the generation of certificate graphs of arbitrary size in a random manner for simulation purposes.

In our public-key management scheme, issuing and revoking certificates are the only operations performed consciously by the users. All the other operations, including authentication, are performed automatically by the nodes, without direct user involvement.

3.2 Certificate Exchange

The *certificate exchange* mechanism is an important and low-cost mechanism that allows nodes to share and distribute certificates that they issue and hold.

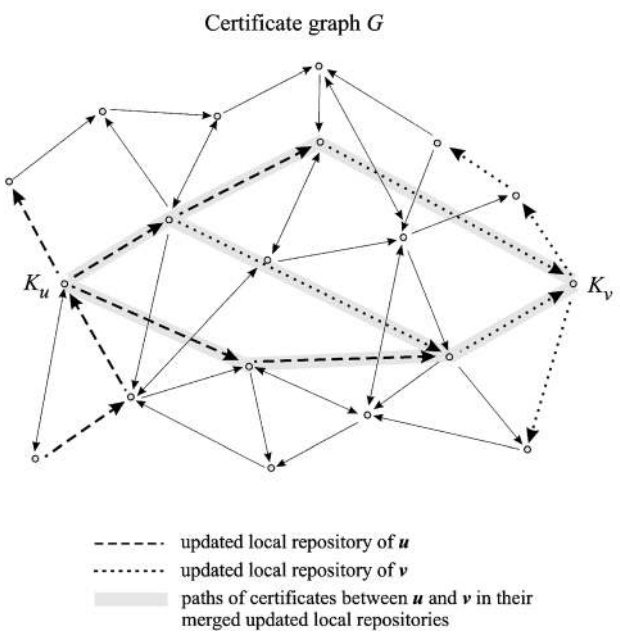

Fig. 2. A certificate graph and paths of certificates between users u and v in their merged updated local repositories.

In an initial phase of the system, each user holds in her local repository only the certificates that she issued and the certificates that other users issued to her. Here, we assume that each time a user u issues a certificate that binds another user v to her public-key K_v, u sends the certificate to v. In this way, each certificate is stored at least twice: by its issuer and by the user to whom it is issued.

The certificate exchange mechanism consists of the periodic exchange of certificates between neighboring nodes. Each node has a local time counter and periodically polls its physical neighbors for certificates. For each node, there is a predefined frequency at which it performs the certificate exchange with its neighbors. This frequency is defined in terms of the exchange period T_E as $1/T_E$. For simplicity, we assume that each node exchanges certificates with the same exchange period T_E. We also note that the nodes do not run the exchange synchronously. The certificate exchange is performed in the following way. Each node u multicasts its subgraphs G_u and G_u^N to its physical neighbors. In this message, u does not send actual certificates but only appropriate unique identifiers (e.g., their hash values). The neighbors of node u that receive the message from u reply with the hash values of the certificates in their updated and nonupdated repositories. Node u then crosschecks the received values with the certificates that it holds and requests from its neighbors only the certificates that it does not hold. Here, it is important to note that when performing the certificate exchange, nodes do not attempt to gather the updates of the certificates that they already hold. The nodes will gather only certificates with different user-key bindings or signatories than those that they already stored. For this, certificates are hashed without their issuing and expiration times.

Only if the local storage of the node becomes too small to store additional certificates, will the node remove expired certificates based on their expiration time (i.e., certificates with the earliest expiration time will be deleted first).

By the certificate exchange mechanism, nodes accumulate certificates in their nonupdated certificate repositories at a low communication cost because the exchanges are performed locally in one hop. As we show later in Section 4.6, after a very short convergence time, the nonupdated certificate repositories of the nodes contain almost the whole certificate graph G. This naturally happens in systems in which certificates are exchanged at a rate higher than certificate creation. After the initial convergence phase, where all certificates are stored by all users, nodes exchange only newly created certificates, but not the updates of the previously stored certificates. An important measure of the certificate exchange is its convergence time T_{CE}, that is the expected time after which, when issued, a certificate reaches all the nodes in the network.

3.3 Constructing Updated Certificate Repositories

The certificate exchange described in the previous section provides the nodes with an incomplete view of the certificate graph and enables them to create their nonupdated certificate repositories. Here, we describe the mechanism by which nodes construct their updated certificate repositories.

Constructing an updated certificate repository of node u means, in terms of our model, selecting a subgraph G_u of the certificate graph G. We assume that each node uses the same local repository construction algorithm to construct its subgraph. When the algorithm is executed on G by node u, it results in a subgraph G_u.

The updated local repository of node u can be constructed in two ways. In the first approach, node u applies the algorithm A on G_u^N, which results in G_u. While executing the algorithm, u checks, by communicating with its issuers, the validity of each certificate that it stores in G_u. In the second approach, node u constructs its updated repository by communicating with its certificate graph neighbors. Typically, these algorithms run in steps where in each step the node decides which certificates to store and how to proceed with the exploration of the certificate graph. In the second approach, nodes construct their updated repositories by exploring only a part of the certificate graph G. As we will see from the example shown in Section 4.5, the nodes do so at a minimal communication cost.

3.4 Certificate Revocation

Each user can revoke a certificate that she issued if she believes that the user-key binding expressed in that certificate is no longer valid. Moreover, if a user believes that her own private key is compromised, she can revoke its corresponding public key. We propose two certificate revocation schemes: explicit and implicit.

In the **explicit revocation** scheme, to revoke a certificate that she issued, the user issues an explicit revocation statement. Due to the way the nodes construct their updated repositories, each node has a list of nodes that request updates for the certificates that it issued. Therefore, when the user revokes a certificate, it does not need to send the revocation to all nodes, but only to the nodes that

regularly update it. Thanks to the certificate exchange scheme, this certificate revocation will reach other nodes as well, but with a delay of the certificate exchange convergence time T_{CE}.

The **implicit revocation** scheme is based on the expiration time of the certificates. Specifically, each certificate is implicitly revoked after its expiration time. As we already described, each certificate contains its issuing time and a validity period T_V. After this period elapses, the certificate is not considered valid anymore. For this reason, it is very important to correctly assign the length of T_V. We assume that a node is able to establish communication with any certificate issuer at some time within a certificate validity period. If this assumption is true, this guarantees that certificate updates will be exchanged regularly. A typical example of a certificate validity period is several days. We suppose that within this period, nodes will be able to update their certificate repositories. If within the given period a node is not able to update some of the certificates in its updated local repository, the node can reconstruct it by using only the certificates available for update. It is important to note that although T_{CE} depends on the network properties, the value for T_V can be flexibly assigned. The value of T_V needs to be chosen carefully to allow nodes to update their repositories.

The described revocation schemes enable nodes to know the status of the certificates in their updated certificate repositories and to be informed, with some delay, of the revocation of other certificates. Furthermore, these schemes allow each user to react to any detected misbehavior by issuing a revocation statement. In addition, key revocation enables users to perform authentication with a higher confidence in the validity of the certificates, but also in the correctness of the user-key bindings contained in the certificates because of their limited validity period.

Key revocation is based on the same scheme as for certificate revocation: If a user believes that her private key has been compromised, she revokes its corresponding public key by notifying the users that issued certificates to her. These users will then use the certificate revocation mechanisms to revoke the certificates that contain the public key in question.

We should also note that the users have strong incentives to maintain their updated certificate repositories, in order to provide sufficient proof of the authenticity of their public keys to other users and to be able to correctly authenticate other keys.

3.5 Coping with Misbehaving Users

A dishonest user may try to trick other users into believing in a false user-key binding by issuing false certificates. She may issue several types of false certificates. First, she may issue a certificate that binds a key K_v to a user f instead of to user v. In this way, a dishonest user may trick other users to believe that K_v is the public key of user f, when it is really the public key of user v. Second, she may issue a certificate that binds user v to a false key K'_v, which may then cause other users to believe that K'_v is indeed the key of user v. Third, a malicious user can invent a number of user names and public keys and bind them by appropriate certificates [8]. The malicious user can then use these public keys to issue false certificates and try to convince a given user that the certificates are correct, as they were signed by many other users. As we will see, our solution prevents these attacks by allowing nodes to detect inconsistent certificates and to determine which user-key bindings are correct.

The certificate exchange mechanism allows nodes to gather virtually all certificates from G. This enables nodes to cross-check user-key bindings in certificates that they hold and to detect any inconsistencies (i.e., conflicting certificates). Two certificates are considered to be conflicting if they contain inconsistent user-key bindings (i.e., if both certificates contain the same username but different public-keys, or if they contain the same public-key, but are bound to different usernames).

If a certificate received by a node u contains a user-key binding (v, K_v) not contained in any certificate in the updated and nonupdated certificate repositories of u, then (v, K_v) and the certificates that certify it are labeled by u as *unspecified*. A certificate labeled unspecified means that the node does not have enough information to assess whether the user-key binding in the certificate is correct. From the moment that (v, K_v) is received, u waits for a predefined period T_P. If within this period u does not receive any conflicting certificates regarding (v, K_v), the status of this binding and of the certificate that certifies it changes to *nonconflicting*. Here, we note that T_P needs to be longer than the expected certificate exchange convergence time T_{CE}. If indeed $T_P > T_{CE}$, nodes will detect inconsistent certificates for all users that exist in the network. For this, each node initially issues a self-signed certificate and exchanges it with other nodes by the certificate exchange mechanism. Thus, the waiting period T_P is actually the expected time for any self-signed certificate to reach all the nodes in the network. However, this mechanism does not prevent users from creating virtual identities or from stealing the identity of people that do not participate in the network.

If a certificate received by a node u contains a user-key binding (v, K_v) that conflicts with a user-key binding (v, K'_v) contained in another certificate held by u, both bindings (v, K_v) and (v, K'_v) and the certificates that certified them are labeled *conflicting*. When a node u detects a conflict, it checks the validity of the conflicting certificates with their issuers and, if they are still valid, tries to resolve the conflict. To resolve the conflict, u tries to find chains of nonconflicting and valid certificates to public-keys K_v and K'_v. Based on the characteristics of the certificate paths (i.e., their number and length), two confidence values that show the user's confidence in the correctness of the two bindings are computed. The two values are then compared and one user-key binding is labeled nonconflicting and the other is labeled false. If, based on the computed confidence values, no decision can be made with respect to the user-key bindings in question, these bindings are labeled as conflicting and the node waits until more information is gathered so that the conflict can be resolved (e.g., if the node receives an additional set of certificates that resolve the conflict, or if two users meet physically, which guarantees that the conflict will be resolved). As discussed in [28], there are several approaches for computing the confidence in the authenticity of a given key. In this paper, we do not detail these approaches, but we simply point out that these values

can be computed and compared automatically by the devices without conscious user involvement.

The presented conflict resolution mechanism can be further used to evaluate the trust in users to issue correct certificates and to detect malicious users.

3.6 Authentication with Helper Nodes

In the introduction, we already described how the key authentication mechanism works. Here, we propose a natural extension to this mechanism. As we already described, when two users want to perform authentication, they merge only their updated certificate repositories. In order to facilitate authentication, a simple extension to the proposed key authentication scheme is for the node u that performs authentication, to take advantage of the certificates from the updated local repositories of the nodes in its physical neighborhood. In this case, we refer to the nodes in the one-hop physical neighborhood of u as the *helper nodes*. The benefits of this extension will be shown in the results of our simulations.

3.7 Load Balancing

We described earlier that, in order to update their certificate repositories, nodes contact the issuers of the certificates that they stored. This approach is not efficient, as the number of nodes that require an update can be high for a given certificate issuer to handle. Even if each node could handle a large number of certificate update requests, it would always not be available to provide the certificates that other nodes need.

Here, we propose a simple load-balancing scheme and we later demonstrate by simulations that this scheme significantly improves the distribution of the communication load among nodes.

Our scheme works as follows: Each node u provides the updates directly to up to s other nodes, and any additional node that needs certificate updates from u gets them from the nodes that get the updates directly from u. Here, s is the size of u's updated local repository. The first s nodes[2] that request updates from u get them directly from u. Each following node that requests a certificate or its update from u receives a list of nodes that get the updates directly from u. Each time that an indirectly updated node requires a certificate update from u, it randomly selects a node from u's list of directly updated nodes and requests an update from that node.

This scheme does not guarantee a perfectly balanced communication load, but shows a simple way to enable nodes to distribute their load to other nodes. In addition, this scheme makes the distribution of certificate updates more robust to node failures and network partitioning.

Unequal load balancing is a consequence of the repository construction algorithm and the characteristics of the certificate graph. More precisely, if the repository construction algorithm is designed such that many nodes keep updated copies of the certificates issued by a single node u, this will increase u's communication load. Such a high communication load of a single node can be due to the specific topology of the certificate graph. In Section 5, we formalize these observations and we explore the mutual dependence of the algorithm performance and the communication load of the node.

More sophisticated solutions can be proposed to solve the load-balancing problem. We do not explore these as load balancing is not the main focus of our work.

2. As a reasonable design decision, we have set the size s of the updated repository of a node to be equal to the number of the nodes to which the node will directly send certificate updates.

4 Simulation Results

The purpose of our simulations is to show, among others, the performance of the certificate repository construction algorithms and the communication cost of the proposed scheme. In this section, we will define several figures of merit, describe the simulation scenarios, and provide the simulation results.

4.1 Algorithm Performance

We define two values that we use to evaluate the performance of our local repository construction algorithms: the basic performance and the shortest path performance.

We define the *basic performance* $p_b(A, s, G)$ of the local repository construction algorithm A with updated repository size s on the certificate graph G as the ratio between the number of key pairs (K_u, K_v) for which there is a directed path from K_u to K_v in their merged subgraphs, and the number of key pairs (K_u, K_v) for which there is a directed path from K_u to K_v in the certificate graph G. Formally, the basic performance is defined as follows:

$$p_b(A, s, G) = \frac{|\{(K_u, K_v) \in V \times V : K_u \rightarrow_{G_u \cup G_v} K_v\}|}{|\{(K_u, K_v) \in V \times V : K_u \rightarrow_{G} K_v\}|}.$$

Therefore, $p_b(A, s, G)$ expresses the fraction of existing directed paths in G that can be reconstructed if only subgraphs G_u and G_v are available.

In a similar way, we define the *shortest path performance*:

$$p_{sp}(A, s, G) = \frac{1}{|W'|} \sum_{(K_u, K_v) \in W} \frac{sp(K_u, K_v, G)}{sp(K_u, K_v, G_u \cup G_v)}, \quad (1)$$

where

$$W' = \{(K_u, K_v) \in V \times V : K_u \rightarrow_{G_u \cup G_v} K_v\},$$

and $sp(K_u, K_v, G)$ is the length of the shortest path between K_u and K_v in G, respectively. In our analysis, we assume that all the subgraphs (updated local repositories) have the same size s.

If the performance values are close to one, it means that our scheme provides essentially the same service as if the whole certificate graph were available to each node.

If authentication is performed with helper nodes (Section 3.6), we denote the performance values by $p_b^h(A, s, G)$ and $p_{sp}^h(A, s, G)$, respectively, where h is the number of helper nodes.

4.2 Key Usage

The *usage* $U_{G,A}(K_v)$ of a given key K_v is defined as:

$$U_{G,A}(K_v) = |\{K_u \in V : K_v \in V(G_u)\}|, \quad (2)$$

where $V(G_u)$ denotes the set of vertices of the subgraph G_u of u.

The key usage of a user is the number of times that her key appears in the updated certificate repositories of other users. This value indicates how many times the user's key could be used for authentication (i.e., could be on the certificate chain that is used for authentication). This value is also a good estimate of the number of times that the certificates issued by a given user are stored by other users in their updated certificate repositories. In both our analytical and simulation results, we show the relation between key usage and performance.

4.3 Communication Cost

The communication cost of our public-key management scheme consists of two parts: the cost of maintaining the updated local repositories and the key authentication cost. These two costs are the consequence of the certificate update and key authentication mechanisms described in Section 3.

The communication cost of the local repository update is directly proportional to the number s of certificates in the local repositories. Upon each update, each node sends s certificate update requests and receives as many responses. The total communication cost of updating the local repositories for all nodes is then

$$\sum_{u \in V} (cert\_size \cdot s_u + cert\_req\_size \cdot s_u) \cdot avg\_num\_hops,$$

where $cert\_size$ is the size in bytes of a certificate, $cert\_req\_size$ is the size of the certificate query, s_u is the size of the local repository of node u, and avg_num_hops is the average number of hops between the nodes in the network.

The cost of updating the local repositories is not equally distributed over the nodes. Clearly, the nodes whose certificates are stored by many nodes will receive a large number of certificate requests and will send the certificate updates to many nodes. We compute the maximum cost of updating the local repositories for a single node as the sum of the cost of sending its own update requests and the cost of responding to update requests of other nodes:

$$\max_{u \in V} [cert\_req\_size \cdot s_u + cert\_size \cdot L(K_u)],$$

where $L(K_u)$ is the load of the owner of K_u, meaning the number of certificates (or certificate updates) requested by other nodes from u. We should note here that a perfectly balanced load means that $L(K_u) = s_u = s$, $\forall u \in V$. In our simulations, we observe the maximum load $L_{max} = \max_{u \in V} L(K_u)$ of a node.

The authentication of K_v by u requires the following message exchange. Node u initiates authentication and requests from v the list of hashes of certificates located in the updated local repository of v. When v replies with the list, u requests the certificates that it needs to complete the authentication; node u requests from v only those certificates that it needs to reconstruct the shortest certificate chain to K_v in $G_u \cup G_v$. In our simulations, we therefore observe, over all pairs of nodes, the average number A of certificates that one node needs to obtain from another node to successfully authenticate its public key.

4.4 Convergence of Certificate Exchange

Here, we observe the benefits of the certificate exchange mechanism (described in Section 3.2) for authentication. For this purpose, we observe two values: *certificate exchange convergence* $CE(t)$ and *user reachability* $UR(t)$. $CE(t)$ is the average (over all users) fraction of certificates from G contained in the nonupdated repository of a user at time t. This value shows how quickly certificates are exchanged between nodes and the time needed for all certificates from G to become a part of the nonupdated repositories of users. $UR(t)$ is the average (over all users) fraction of keys to which a user can find a path within her updated and nonupdated repositories at time t. The user reachability shows the usefulness of the certificate exchange scheme for the authentication through the nonupdated repositories. Both values depend on the mobility of the nodes and on the certificate exchange period T_{CE}.

4.5 Simulation Scenario

We now describe the simulation scenario where we observe our public-key management scheme. We use the Maximum Degree construction algorithm for the construction of the updated repositories. The algorithm is described below. We test its performance on PGP, and on random and artificial certificate graphs (created according to the model proposed in [14]). The mobility model is the random waypoint model (described hereafter). All simulations were performed with a simulator implemented in C/C++ with LEDA (Library of Efficient Data Types and Algorithms) [29].

4.5.1 Maximum Degree Algorithm

The *Maximum Degree* algorithm selects a subgraph that consists of two logically distinct parts: an *out-bound* and an *in-bound* subgraph. More precisely, the subgraph consists of several vertex-disjoint out-bound and vertex-disjoint in-bound paths (the subgraph that is built resembles a star). When starting from a vertex K_u, Maximum Degree builds $e_{out} = \min(deg_{out}, c)$ vertex-disjoint out-bound and $e_{in} = \min(deg_{in}, c)$ vertex-disjoint in-bound paths where deg_{out} and deg_{in} denote the number of K_u's outgoing and incoming edges, respectively, and c is a predefined constant that represents the desired number of paths to be built. The lengths ℓ_{in} and ℓ_{out} of in-bound and out-bound paths are computed as $\lceil s/2e_{in} \rceil$ and $\lceil s/2e_{out} \rceil$, respectively, where s is an input of the algorithm, representing the required number of vertices of the resulting subgraph.

The algorithm runs in multiple rounds. In the first round, the algorithm starts from the selected vertex K_u (the public key of the user constructing the subgraph) and includes in its out-bound subgraph e_{out} outgoing edges (with its terminating vertices) that originate from K_u. We denote the set of the destination vertices of those edges by D_{out}. In each following step, e_{out} edges (and their terminating vertices) are selected so that they originate from the vertices in the set D_{out}, but in a way that no two edges have the same originating vertex or lead to the same destination vertex (this ensures that the paths are disjoint). In each step, the vertices in D_{out} are replaced with the last set of terminating vertices. In practice, this means that node u must ask the nodes belonging to the vertices in D_{out} for the list of their outgoing edges. This list can easily be provided because each node stores its outgoing edges (the certificates that it issued).

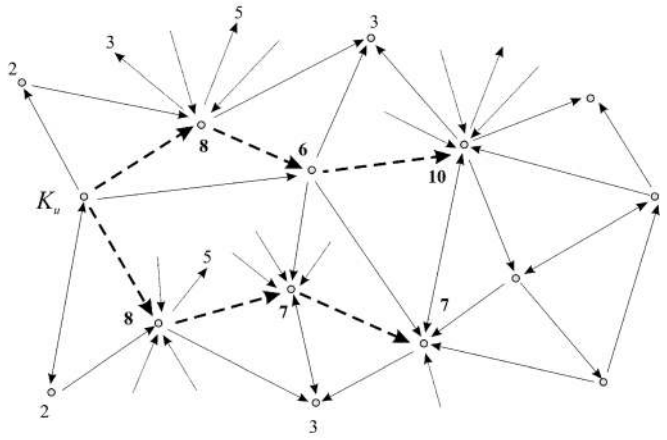

Fig. 3. An example of out-bound subgraph of vertex K_u (represented in dashed lines), constructed with the Maximum Degree algorithm with $c = 2$ and $s = 6$.

The construction of the in-bound subgraph of u is similar: The algorithm starts from vertex K_u, and in the first round, e_{in} incoming edges (with their originating vertices) that terminate in K_u are selected into the in-bound subgraph. The originating vertices are stored in the list D_{in}. In each following step, e_{in} edges are added to the subgraph that have destinations in D_{in}, so that no two edges have common originating or common destination vertices. D_{in} is then updated with the set of the originating vertices of the edges that are selected by the algorithm. In order to ensure that the in-bound subgraphs can be built, each user must know both its outgoing and its incoming edges. For this reason, our algorithm requires that each user is notified whenever another user issues her a certificate.

The selection of the edges and their terminating or originating vertices, in each round of the algorithm, is based on their degree. More precisely, in each step, vertices that have the highest degree are selected. Therefore, only local knowledge (the neighbors degrees) is necessary for the nodes to perform the Maximum Degree algorithm. An example of an out-bound subgraph constructed with the Maximum Degree algorithm is shown in Fig. 3.

4.5.2 Random Waypoint Mobility Model

In the random waypoint mobility model [30], a mobile node moves on a finite continuous plane from its current position to a new location by randomly choosing its destination coordinates, its speed of movement, and the time that it will pause when it reaches the destination. After the pause time, the node chooses a new destination, speed, and pause time. This is repeated for each node, until the end of the simulation time.

4.5.3 Certificate Graphs

In our simulations, we use PGP graphs, random graphs, and artificial certificate graphs. PGP graphs are extracted from the PGP database at http://pgp.dtype.org. This database contains the information about public-keys and public-key certificates issued and revoked since the launch of the PGP project in the early 1990's until today. Artificial certificate graphs are created based on the model that we defined in [14]. These graphs are designed in such a way that they exhibit characteristics similar to PGP certificate graphs, notably small-world characteristics. In all three cases, for our simulations, we use either the full graph if it is strongly connected, or its largest strongly connected component, otherwise.

4.6 Results

In this section, we present the main simulation results. In the following figures, the number of vertices and the number of edges of a graph are denoted by $n = |V|$ and $m = |E|$, respectively. Fig. 4 shows the performance of the Maximum Degree algorithm on random, PGP, and artificial certificate graphs of various sizes. We observe that on all types of graphs, Maximum Degree exhibits high performance, even if the size of the updated local repository is small compared to the number of users and the total number of certificates in the certificate graph. Fig. 5 shows that the basic performance can be further increased by using the updated certificate repositories of the helper nodes. This high performance demonstrates that even with a simple algorithm such as Maximum Degree, the users have a high probability of performing authentication by using only their updated local repositories and the repositories of their physical neighbors. Thus, even in a very small network partition where a node

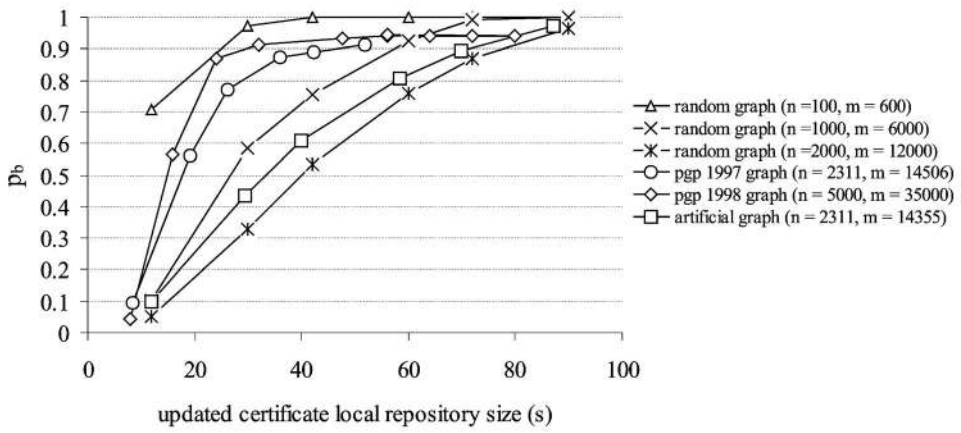

Fig. 4. Basic performance of the Maximum Degree algorithm (three paths) on random, PGP, and artificial certificate graphs for various sizes of the updated certificate repositories.

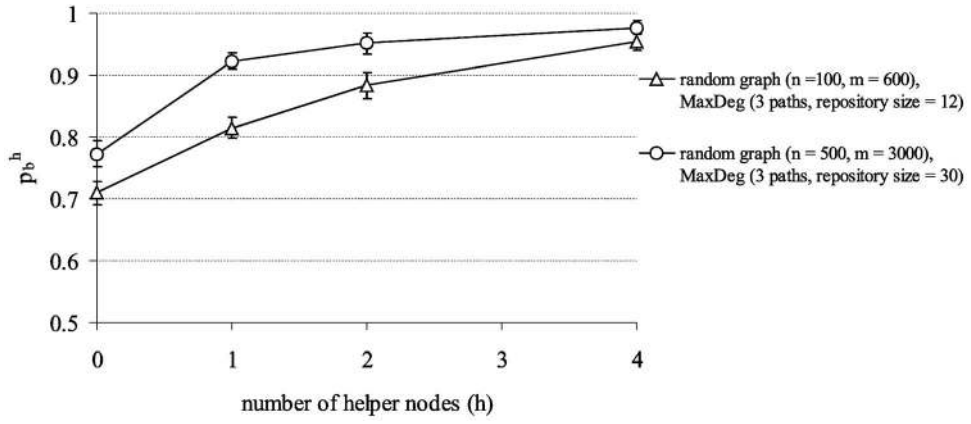

Fig. 5. Basic performance of the Maximum Degree algorithm with helper nodes on random certificate graphs. The results are displayed with the 95% confidence interval.

can communicate with four helper nodes only, there is still a high probability that authentication is possible.

Table 1 shows that the average shortest path in the merged updated certificate repositories is small, which implies a small authentication cost because only a few certificates need to be exchanged between the nodes. From the shortest path performance, we observe that the lengths of the shortest paths in the merged updated repositories are not significantly longer than those in the whole certificate graph. The communication load results show that our load-balancing scheme efficiently limits the nodes' maximum load to the size of their updated certificate repositories.

On Fig. 6, we introduce mobility in order to observe the certificate exchange convergence and the user reachability. We see that, on average, each node collects most of the certificates from the certificate graph after a very short time. This enables nodes to efficiently detect conflicting certificates. This result also provides a good basis for estimating the necessary values of T_P and T_{CE}. Furthermore, user reachability shows that after a short time, nodes have collected enough certificates to find a path to most of the other users through their nonupdated certificate repositories. This shows that even if authentication through updated local repositories fails, there is a very high probability that the node will still be able to perform authentication by finding certificate paths in its nonupdated repository and requesting an update of the necessary certificates.

5 Analytical Results

In this section, we analyze two important problems: 1) Find an efficient repository construction algorithm that minimizes the size of the updated certificate repositories and achieves a predefined basic performance. 2) Find an efficient repository construction algorithm that minimizes the size of the updated certificate repositories, achieves a predefined basic performance and achieves a balanced *key usage*. We note that, as we discussed earlier in Section 4.3, by minimizing the size of the updated certificate repositories, we minimize the overall communication cost of the public-key management system.

More formally, given a certificate graph $G(V, E)$, and $U_0 \in \mathbb{N}$, $p_0 \in [0, 1]$, we consider the following two problems:

TABLE 1
Average Shortest Paths Lengths, Shortest Path Performance p_{sp}, Node's Maximum Communication Load L_{max}, and the Average Number A of Certificates Transferred per Authentication, for Random (n = 500, m = 3,000) and PGP 1996 (n = 1,345, m = 7,025) Certificate Graphs with the Maximum Degree Algorithm

graph; rep. size (paths × length)	avg shortest path length in $G_u \cup G_v$	p_{sp}	node's maximum communication load L_{max}	A
PGP; 24 (6 × 4)	6.87	0.82	24	3.53
Random; 30 (6 × 5)	5.83	0.70	30	2.95
Random; 42 (6 × 7)	6.32	0.66	42	3.42
Random; 30 (6 × 5) 3 helpers	6.30	0.67	30	0.53+2.23

Note that, with helper nodes, the number of transferred certificates includes the certificates obtained from the helpers and some certificates obtained from the other nodes.

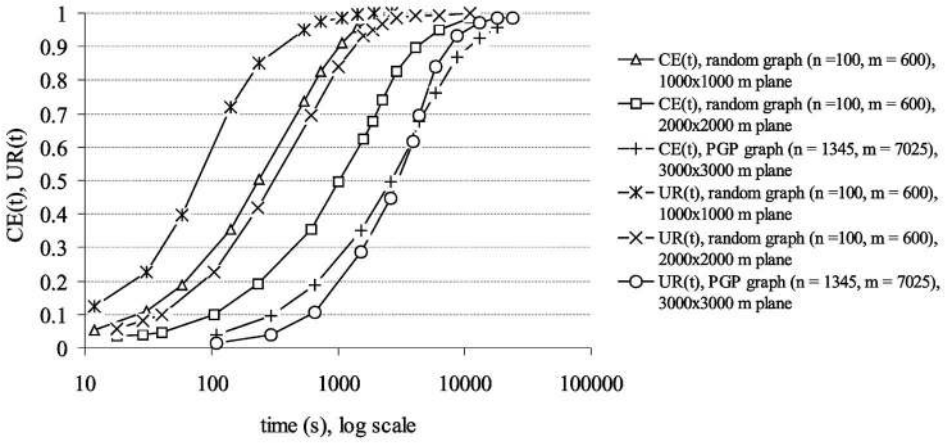

Fig. 6. Certificate exchange convergence $CE(t)$ and user reachability $UR(t)$ with random waypoint model (max. speed = 20m/s, max. pause =20s, certificate exchange period $T_{CE} = 60$s, power range =120m) on random and PGP certificate graphs.

- *Problem 1*: Find a local repository construction algorithm A that **minimizes** the size s of the updated certificate repository, such that $p(A, s, G) \geq p_0$.
- *Problem 2*: Find a local repository construction algorithm A that **minimizes** the size s of the updated certificate repository, such that $p(A, s, G) \geq p_0$ and $max_v U_{G,A}(K_v) \leq U_0$, where $U_{G,A}(K_v)$ is the usage of K_v.

5.1 Problem 1: Minimize the Size of the Updated Certificate Repositories

If $p_0 = 1$, this problem can be stated in the following way: Find a set $\{G_v : K_v \in V\}$ of subgraphs such that for each pair of vertices (K_u, K_v), there exists a path from K_u to K_v in $G_u \cup G_v$, and the size of the largest local repository is minimized.

We obtain an upper bound on the minimal repository size s_{min} by constructing the subgraph G_v of each vertex K_v as the union of the shortest path from K_v to K_x and the shortest path from K_x to K_v, where K_x is the vertex that has the smallest maximal distance to and from all the vertices in V. Thus, the minimal local repository size s_{min} is bounded by

$$s_{min} \leq \min_{K_x \in V} \max_{K_v \in V, K_v \neq K_x} (d(K_x, K_v) + d(K_v, K_x)), \qquad (3)$$

where $d(K_v, K_x)$ is the length of the shortest path between K_v and K_x in G.

The reasoning behind this construction is that it is easy to show that the sizes of local repositories increase if the users store the shortest paths to (and from) more than one vertex.

This construction shows that it is possible to achieve, with a simple repository construction algorithm, a very high performance with small updated certificate repositories and therefore at a low communication cost. This conclusion is based on the characteristics of the certificate graphs. For the PGP, small-world, and random graphs, the average shortest paths are $\sim \log |V|$. This shows that for the majority of the users, the size of the updated certificate repositories in these certificate graphs will be $\sim \log |V|$.

5.2 Problem 2: Minimize the Size of the Local Repositories and the Key Usage

Now, we turn our attention to the problem of minimizing the size of the updated certificate repositories if we limit the maximum key usage. Here, we show that under certain conditions, it is possible to compute a lower bound on the size of the certificate repositories.

In the following, we will assume that the subgraph construction algorithm A generates a subgraph G_u for u, which consists of an in-bound subgraph and an out-bound subgraph that we denote by G_u^{in} and G_u^{out}, respectively, with the following properties:

- For all $K_v \in V(G_u^{in})$, there exists a path from K_v to K_u in G_u^{in}, and
- For all $K_v \in V(G_u^{out})$, there exists a path from K_u to K_v in G_u^{out},

where $V(G_u^{in})$ and $V(G_u^{out})$ denote the set of vertices of G_u^{in} and G_u^{out}, respectively. Note that $V(G_u^{in})$ and $V(G_u^{out})$ are not necessarily disjunct.

The following theorem states that if 1) the desired algorithm performance is 1, 2) each key should have the same usage, and 3) the subgraphs of the users have the same number of vertices, then the size of the local certificate repositories of the users (the number of certificates stored) cannot be smaller than $\sqrt{|V|} - 1$, where $|V|$ is the total number of users in the system.

Theorem 1. *Let us consider a certificate graph $G(V, E)$ and a subgraph construction algorithm A which selects an in-bound and an out-bound subgraph for each user as described above. If 1) $p(A, s, G) = 1$, 2) $U_{G,A}(K_u) = U_{G,A}(K_v)$ for all K_u, $K_v \in V$, and 3) $|V(G_u)| = s$ for each $K_u \in V$, then $s \geq \sqrt{|V|} - 1$.*

Proof. Let us consider an arbitrary vertex K_w of the certificate graph. Let us denote the set of users (vertices) that stored K_w in their in-bound subgraph by S_1. Formally, S_1 is defined as follows:

$$S_1 = \{K_z \in V : K_w \in V(G_z^{in})\}. \qquad (4)$$

Furthermore, let us denote by S_2 the set of users (vertices) that stored the vertices in $V(G_w^{out})$ in their inbound subgraphs. Formally:

$$S_2 = \{K_z \in V : \exists K_{z'} \in V(G_w^{out}) : K_{z'} \in V(G_z^{in})\}. \quad (5)$$

Since $p(A, s, G) = 1$, there must be a path from K_w to K_z in $G_w \cup G_z$ for every $K_z \in V$. This means that K_z is either in S_1, or in $V(G_w^{out})$, or in S_2. In other words, the following must hold:

$$S_1 \cup V(G_w^{out}) \cup S_2 = V \setminus \{K_w\}. \quad (6)$$

Since $|V(G_u)| = s$ for all $K_u \in V$, it trivially follows that $|V(G_w^{out})| \leq s$. In addition, since $U_{G,A}(K_u) = U_{G,A}(K_v)$ for all $K_u,\ K_v \in V$, and $|V(G_u)| = s$ for all $K_u \in V$, it is easy to see that $U_{G,A}(K_u) = s$ must hold for all $K_u \in V$. This means that $|S_1| \leq s$ and $|S_2| \leq s^2$ must hold. Now, we get the following:

$$\begin{aligned} |V| - 1 &= |V \setminus \{K_w\}| \\ &= |S_1 \cup V(G_w^{out}) \cup S_2| \\ &\leq |S_1| + |V(G_w^{out})| + |S_2| \\ &\leq s^2 + 2s. \end{aligned}$$

From this, the statement of the theorem follows directly. □

We do not know how sharp this lower bound is, as we were not able to find any construction that satisfies the conditions of Theorem 1 and achieves strict equality. We found a construction that satisfies the conditions of Theorem 1 and achieves a subgraph size of $2 \cdot (\sqrt{|V|} - 1)$, assuming that $G_w^{out} = G_v^{in},\ \forall K_w, K_v \in V$, but for lack of space, we will present it in our future reports.

Theorem 1 shows that the requirement of an equal key usage is a severe design criterion, meaning that if it is respected, then the size of the updated certificate repositories, and therefore the communication costs, must be high. On the other hand, the $\log|V|$ bound on the minimal updated repository size shows that the communication cost can be very low if a balanced key usage is not an issue. As we have shown earlier, a simple load-balancing scheme can alleviate the problem of high key usage and provide the nodes with high performance at a low and balanced communication cost.

6 Conclusion

In this paper, we addressed the difficult problem of key management in mobile ad hoc networks. We proposed a fully self-organized public-key management scheme that does not rely on any trusted authority or fixed server, not even in the initialization phase. To the best of our knowledge, our proposal is the first in which public-key management is fully self-organized. In our approach, each user is her own authority and issues public-key certificates to other users. Certificates are stored and distributed by the nodes and each node maintains a local certificate repository that contains a limited number of certificates selected by the node according to an appropriate algorithm. Key authentication is performed via chains of certificates. When user u wants to verify the authenticity of the public key of another user v, they merge their local certificate repositories and u evaluates the authenticity of K_v based on the certificates contained in the merged repository. The detection of false certificates is enabled through the certificate exchange scheme that allows nodes to detect any conflicting certificates.

The main contributions of this work can be summarized as follows:

1. We proposed a fully self-organized public-key management system for mobile ad hoc networks.
2. We showed that two users in a mobile ad hoc network can perform key authentication based only on their local information, even if security is performed in a self-organized way.
3. We showed that with a simple local repository construction algorithm and a small communication overhead, our system achieves high performance on a wide range of certificate graphs.
4. We demonstrated that nodes can exploit mobility to facilitate authentication and to detect inconsistent and false certificates.

This public-key management scheme is designed primarily for use in mobile ad hoc networks. For this reason, our repository construction algorithm assumes only partial knowledge of the certificate graph and is designed with the communication overhead in mind. An important feature of this scheme is that key authentication is still possible even when the network is partitioned and nodes can communicate with only a subset of other nodes. It is also important to note that the proposed solution requires users' conscious involvement only when their public/private key pairs are created and for issuing and revoking certificates; all other operations (including certificate exchange and construction of certificate repositories) are fully automatic.

Our future work includes further exploration of mechanisms for the detection and the resolution of inconsistent certificates, improvement of the certificate graph models, and exploration of more sophisticated load-balancing/data management schemes (e.g., [31]) for public-key management in mobile ad hoc networks.

Acknowledgments

The authors are thankful to Ivan Mitev, Mario Cagalj, Serge Vaudenay, and Dominique de Werra for their contribution to this work. The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 (http://www.terminodes.org).

References

[1] C.E. Perkins, *Ad Hoc Networking*. Addison Wesley Professional, Dec. 2000.
[2] D.B. Johnson, "Routing in Ad Hoc Networks of Mobile Hosts," *Proc. IEEE Workshop Mobile Computing Systems and Applications*, Dec. 1994.
[3] J. Jubin and J.D. Turnow, "The DARPA Packet Radio Project" *Proc. IEEE*, 1987.

[4] L.M. Kornfelder, "Toward a Practical Public-Key Cryptosystem," bachelor's thesis, Dept. Electrical Eng., Massachusetts Inst. of Technology, Cambridge, 1978.
[5] P. Zimmermann, *The Official PGP User's Guide.* MIT Press, 1995.
[6] L. Zhou and Z. Haas, "Securing Ad Hoc Networks," *IEEE Network,* vol. 13, no. 6, pp. 24-30, Nov./Dec. 1999.
[7] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang, "Providing Robust and Ubiquitous Security Support for Mobile Ad Hoc Networks," *Proc. Ninth Int'l Conf. Network Protocols (ICNP),* Nov. 2001.
[8] J. Douceur, "The Sybil Attack," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2002.
[9] G. Montenegro and C. Castelluccia, "Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses," *Proc. Ninth Ann. Network and Distributed System Security Symp. (NDSS),* 2002.
[10] G. O'Shea and M. Roe, "Child-Proof Authentication for MIPv6 (CAM)," *ACM Computer Comm. Rev.,* Apr. 2001.
[11] J.-P. Hubaux, T. Gross, J.-Y. Le Boudec, and M. Vetterli, "Toward Self-Organized Mobile Ad Hoc Networks: The Terminodes Project," *IEEE Comm. Magazine,* Jan. 2001.
[12] L. Blazevic, L. Buttyan, S. Capkun, S. Giordano, J.-P. Hubaux, and J.-Y. Le Boudec, "Self-Organization in Mobile Ad Hoc Networks: The Approach of Terminodes," *IEEE Comm. Magazine,* June 2001.
[13] J.-P. Hubaux, L. Buttyan, and S. Capkun, "The Quest for Security in Mobile Ad Hoc Networks," *Proc. ACM Symp. Mobile Ad Hoc Networking and Computing (MobiHoc),* 2001.
[14] S. Capkun, L. Buttyan, and J.-P. Hubaux, "Small Worlds in Security Systems: an Analysis of the PGP Certificate Graph," *Proc. ACM New Security Paradigm Workshop (NSPW),* 2002.
[15] L. Buttyan and J.-P. Hubaux, "Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks," to appear in *ACM/Kluwer Mobile Networks and Applications (MONET),* vol. 8, no. 5, Oct. 2003.
[16] F. Stajano and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-Hoc Wireless Networks," *Proc. Seventh Int'l Workshop Security Protocols,* 1999.
[17] F. Stajano, *Security for Ubiquitous Computing.* John Wiley and Sons, Feb. 2002.
[18] R. Anderson and M. Kuhn, "Tamper Resistance—A Cautionary Note," *Proc. Second Usenix Workshop Electronic Commerce,* 1996.
[19] A. Pfitzmann, B. Pfitzmann, and M. Waidner, "Trusting Mobile User Devices and Security Modules," *Computer,* Feb. 1997.
[20] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," *Proc. ACM Int'l Conf. Mobile Computing and Networking (MobiCom),* 2000.
[21] N. Asokan and P. Ginzboorg, "Key Agreement in Ad Hoc Networks," *Computer Comm.,* vol. 23, pp. 1627-1637, 2000.
[22] M. Guerrero Zapata and N. Asokan, "Securing Ad Hoc Routing Protocols," *Proc. ACM Workshop Wireless Security (WiSe),* Sept. 2002.
[23] K. Sanzgiri, B. Dahill, B.N. Levine, C. Shields, E.M. Belding-Royer, "A Secure Routing Protocol for Ad Hoc Networks," *Proc. Int'l Conf. Network Protocols (ICNP),* Nov. 2002.
[24] Y.-C. Hu, D.B. Johnson, and A. Perrig, "SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks," *Proc. Fourth IEEE Workshop Mobile Computing Systems and Applications,* June 2002.
[25] S. Milgram, "The Small World Problem," *Psychology Today,* vol. 61, 1967.
[26] J. Travers and S. Milgram, "An Experimental Study of the Small World Problem," *Sociometry,* vol. 32, 1969.
[27] D. Watts, *Small Worlds.* Princeton Univ. Press, 1999.
[28] M. Reiter and S. Stubblebine, "Authentication Metric Analysis and Design," *ACM Trans. Information and System Security,* vol. 2, no. 2, pp. 138-158, 1999.
[29] Algorithmic Solutions, "LEDA," www.algorithmic-solutions.com, 2003.
[30] T. Camp, J. Boleng, and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Comm. and Mobile Computing (WCMC): Special Issue on Mobile Ad Hoc Networking: Research, Trends, and Applications,* vol. 2, no. 5, pp. 483-502, 2002.
[31] J. Luo, P.T. Eugster, and J.-P. Hubaux, "Route Driven Gossip: Probabilistic Reliable Multicast in Ad Hoc Networks," *Proc. INFOCOM 2003,* 2003.

Srdjan Capkun received the BSc degree in electrical engineering/computer science from the University of Split, Croatia, in 1998. In 1999, he joined the Doctoral School in communication systems in the Department of Communication Systems at the Swiss Federal Institute of Technology—Lausanne (EPFL). In September 2000, he joined the Laboratory for Computer Communications and Applications (LCA) at EPFL, where he is working toward his PhD degree. His current research interests include security and positioning issues in mobile ad hoc networks. He is a student member of the IEEE.

Levente Buttyán received the MSc degree in computer science from the Budapest University of Technology and Economics (BUTE) in 1995 and the PhD degree from the Swiss Federal Institute of Technology—Lausanne (EPFL) in 2002. His dissertation is concerned with formal models for authenticated key transport and rational exchange protocols. From 1997 to 2002, Dr. Buttyan worked in the group of Professor Jean-Pierre Hubaux in the Laboratory of Computer Communications and Applications at EPFL. In 2003, he joined the Department of Telecommunications at BUTE, where he currently holds a position as assitant professor and works in the Laboratory of Cryptography and Systems Security (CrySyS). His research interests include the design of cryptographic protocols, many aspects of network security (including wireless networks), and the problems of authentication and cooperation in ad hoc networks. He is a student member of the IEEE.

Jean-Pierre Hubaux joined the Swiss Federal Institute of Technology—Lausanne (EPFL) as an associate professor in 1990; he was promoted to full professor in 1996. His research activity is focused on mobile networking and computing, with a special interest in fully self-organized mobile ad hoc networks. In particular, he has performed research on cooperation aspects, security, power efficiency, and distributed algorithms for ad hoc networks. During the last few years, he has been strongly involved in the definition and launching phases of a new National Competence Center in Research named "Mobile Information and Communication Systems" (NCCR/MICS), see http://www.terminodes.org. He served as the general chair for the Third ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002), held in June on the campus of EPFL. He is an associate editor of *IEEE Transactions on Mobile Computing* and of the *Elsevier Journal on Ad Hoc Networks*. Within his first year at EPFL, he defined the first curriculum in communication systems. From October 1999 until September 2001, he was the first chairman of the Communication Systems Department. He has held visiting positions at the IBM T.J. Watson Research Center and at the University of California at Berkeley. He has published more than 60 papers in the area of networking. In the past, he spent 10 years in France with Alcatel, where he was involved in R&D activities, mostly in the area of switching systems architecture and software. He is a senior member of the IEEE and a member of the ACM. For more information, check http://lcawww.epfl.ch/hubaux.