Towards a survivable security architecture for ad-hoc networks

Tuomas Aura[1], Silja Mäki[2]

[1] Microsoft Research Ltd.
7 J J Thomson Avenue, Cambridge, CB3 0FB, UK
`tuomaura@microsoft.com`

[2] Helsinki University of Technology, Laboratory for Theoretical Computer Science
P.O.Box 5400, FIN-02015 HUT, Finland
`Silja.Maki@hut.fi`

Abstract. We present a security architecture for access control in ad-hoc networks of mobile electronic devices. Ad-hoc networks are formed on demand without support from pre-existing infrastructure such as central servers, security associations or CAs. Our architecture is fully distributed and based on groups and public-key certification. The goal is a survivable system that functions well even when network nodes fail and connections are only occasional. We identify some open problems in the optimal use of unreliable communications for security management.

1 Introduction

Consider the digital appliances, such as desktop PC, mobile phone, MP3 player, palmtop computer, and car computer, that belong to one person. While each one of these devices is useful alone, their full potential will only be realized when they are connected into a network to share information and services with each other. The equipment must also be able to connect to outside services such as a printer or an email server. Much work is currently done on this kind of ad-hoc networking of smart devices with wireless connections [Blu99,IEE97].

Ad-hoc networks differ considerably from the fixed communications networks. The nodes and connections are inherently unreliable. The networks are created on demand, without support from fixed infrastructure such as central servers and routers. The networks, in particular, security in these environments must be managed without the help of pre-existing trusted servers, naming schemes, certification authorities (CAs), or security associations.

The organization of ad-hoc networks is often based on *groups* of nodes [CJS99,Roe97]. In this paper, we describe an architecture for securely creating groups, managing their membership, and proving group membership. The design is fully distributed, based on key-oriented key certificates, and *survivable in situations where nodes fail and communication is only occasional*.

The primary use of our architecture is in access control but it can also be used for authentication in various group key exchange protocols, e.g. [BD94,AST00,AG00]. In addition to the personal networks of electronics, the protocol may find applications in other systems with similar characteristic, including office and home appliances and mobile rescue and military operations.

Much of the complexity in our design comes from the fact that on-line verification of access rights and predictable distribution of revocation lists cannot be implemented with the completely unreliable communication. We present a best-effort solution that seems like a reasonable compromise and identify open theoretical problems in optimization of the revocation procedure.

2 Distributed group membership management

With a group, we mean a set of members, persons or other physical or logical entities that may collectively be given access rights. Some of the members are leaders and have the right to make decisions about group membership. Groups may also have subgroups.

The membership management is based on public-key certificates. Our view is key-oriented. As in the SPKI [EFL+99b,EFL+99a], SDSI [RL96] and KeyNote [BFIK99] infrastructures, called the group is represented by a public key, the group key, and its members are identified by their respective public keys.

2.1 The purpose of groups

The primary purpose of a group is that it may be granted access rights, which any member of the group is entitled to use. For example, the group members could be allowed to use a printer or to access a database. The group membership, as we define it, never imposes any obligations onto the members. It follows that the members of a group (e.g. printer users) do not need to trust each other in any respect. In fact, they might not even know about each other.

However, a common use for a group is co-operation and mutual authentication inside the group. For example, the electronic appliances owned by a single person may be configured to have full access to each other. The group members may create a shared encryption key for protecting communication between themselves. Every time a member communicates with another one and transmits or receives a message encrypted or authenticated with the shared key, it makes the implicit decision to trust the group with its secrets or its integrity. The group members may also decide to provide each other services like message routing and network management. In every case, the members must have a reason, such as an administrative decision, for trusting the specific group with respect to the purpose for which they use it. It is a prudent practice to write the original purpose of the group explicitly on all the member certificates.

2.2 The basic group structure

A new group is created by generating a fresh signature key pair. The new key KG is called the *group key*. The public signature key is used as the group identifier. The group key is also used to certify new members and to verify membership. The owner of the group key is called the *leader*. Anyone capable of generating a fresh signature key may start a new group and become its leader. This way, groups can be created ad hoc.

In the beginning, the leader is the only member of the group. The leader may admit other members by issuing *member certificates* to their public keys (*member keys*). The certificates are signed with the group key KG. A member certificate contains the group identifier (i.e. the public group key), the public key of the member, an optional validity period and a signature signed with the group key.

To protect against the loss or disconnection of the leader, the leader may distribute its powers by appointing other leaders. It does this by issuing *leader certificates* to their public keys (*leader keys*). The leader certificates have the same structure as the member certificates. All leaders of the group are also members of it.

All the leaders have an equivalent authority with the original leader and each one may act independently. They all may admit new members and new leaders to the group by signing certificates with their own keys. All the certificates will have the same group identifier (the public group key or its secure hash value) on them.

The multiple independent leaders make the group survivable. If some leaders are removed from the system either temporarily or permanently, the remaining leaders can still continue operation. A disadvantage is that no single leader is guaranteed to have a whole picture of the group membership at a given moment. The leaders should have a protocol for sharing information about the members with each other whenever communication is possible.

The protocol allows the leaders to act independently because any mechanical restrictions will make the system less reliable. In practice, however, the leaders should follow some agreed policy in making their decisions or in taking over leadership when connection to another leader is lost. For instance, in our example of personal appliances, the preferences of the user operating the equipment will in a natural way create such a policy.

We note that while *threshold schemes* [Des94,ZH99] can theoretically improve the security of mobile networks where the nodes are exposed to physical danger, they can easily lead to denial of service and create frustrated users. Hence, they are not suitable for distributed systems where the device owners from consumers to military leaders

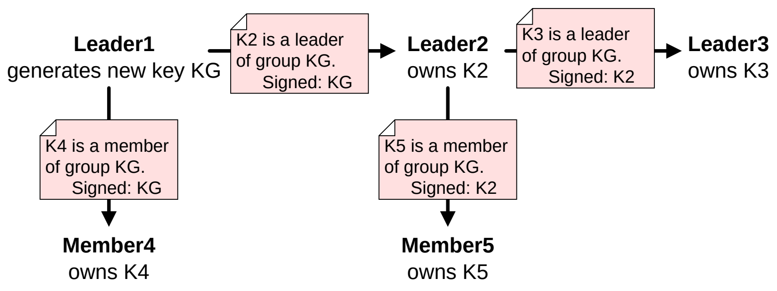
Leader1
generates new key KG
K2 is a leader of group KG. Signed: KG
Leader2
owns K2
K3 is a leader of group KG. Signed: K2
Leader3
owns K3
K4 is a member of group KG. Signed: KG
K5 is a member of group KG. Signed: K2
Member4
owns K4
Member5
owns K5

Fig. 1. A group with multiple leaders

want to implement their decisions instantly without waiting for approval from several possibly unreachable peers. Therefore, our architecture allows every authorized leader to act independently.

When a leader certifies other leaders or members, it passes along all the certificates that prove its own status as a leader in the group. This way, a *certificate chain* is formed from the group key to each member key. A member can prove its membership to another group member or to an outsider by presenting its certificates and by using its own private key.

Since the group key is the group identifier, everyone doing business with the group will automatically know it. Services, like a printer, will have the group key in an access control list (ACL). The group key from the ACL is used to verify the first certificate in the chain. Each of the following certificates is verified with the leader key certified in the previous certificate. No names or certification authorities are needed in the verification process, and only a single key per group is needed in any ACL. Moreover, the verification of access rights can be implemented so that the verifier uses only a constant amount of memory.

Fig. 1 illustrates a group with multiple leaders. (For simplicity, the validity periods of the certificates are not shown in the figure.) When Member 5 of Fig. 1 joined the group, it received the membership certificate signed by Leader 2 and the leader certificate issued to Leader 2, which is signed by KG. The latter certificate proves the authority of Leader 2. When Member 5 wants to prove its membership, it presents both certificates and proves its possession of the key $K5$.

2.3 Subgroups

Sometimes a leader may want to admit all members of another group to its own group. For this purpose, we have defined a *subgroup certificate*, which any leader can issue to the group key of the subgroup. The subgroup certificate has the same structure as

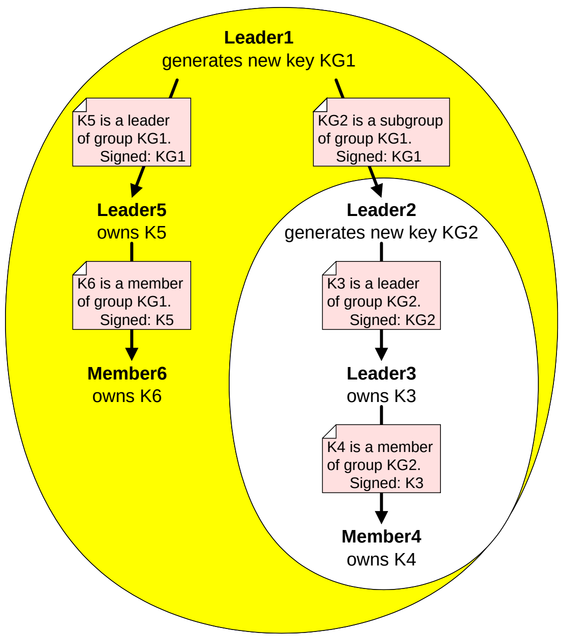
Leader1
generates new key KG1
K5 is a leader of group KG1. Signed: KG1
KG2 is a subgroup of group KG1. Signed: KG1
Leader5
owns K5
K6 is a member of group KG1. Signed: K5
Member6
owns K6
Leader2
generates new key KG2
K3 is a leader of group KG2. Signed: KG2
Leader3
owns K3
K4 is a member of group KG2. Signed: K3
Member4
owns K4

Fig. 2. Subgroups

a member or leader certificate. It contains the *supergroup* and subgroup identifiers (i.e. keys), a validity period, and a signature.

All the members of a subgroup are also members of the supergroup. The leaders of a subgroup may continue to admit new members to the subgroup. Any changes in the subgroup affect also the membership of the supergroup. Otherwise, the subgroup and supergroup are managed independently. The subgroup leaders are not leaders in the supergroup or vice versa, unless they are separately certified as such. The aim is to keep the membership management functions as simple and local as possible.

Figure 2 shows how the group $KG2$ becomes a subgroup of the group $KG1$. The leaders of the groups $KG1$ and $KG2$ can continue to admit new members, but not directly to each other's groups.

The membership of a group is transitive in a hierarchy of subgroups but applications should keep the depth of this hierarchy to the minimum. In fact, we suggest avoiding the complications of subgroups altogether except when they simplify the system management significantly. A typical situation where subgroups are useful is the management of a small hierarchical organization. For example, the electronic appliances of a household include shared items and the personal items of each family member.

3 Revoking membership

When a member voluntarily leaves a group, it should discard the member certificate and, if possible, erase its signature key. In many applications, the members can be trusted to do this. For example, when an electrical device is sold or given away, the owner can be expected to reset its memory.

Nevertheless, there are situations where the members may not be co-operative. If a member stops paying for a service, a device is stolen, or there is a sudden suspicion that a remote computer has been hacked, some preemptive means is needed to remove them from the group.

3.1 Simple methods

The most effective way to get rid of unwanted members or leaders is *group reconstitution*, replacing the group key with a new one and reissuing all certificates. This guarantees that no unwanted members remain in the group. The problem is that the certificates must be distributed and the authentic new group key must be conveyed to all the places where membership is verified. This costs time and may be impossible when communication is unreliable. In practice, the new group has to replace the old one gradually so that any nodes unaware of the new group can still use the old keys and certificates.

Another secure way to get rid of unwanted certificates is to have *short validity times* and to require the members to renew their membership regularly. This means that all the members must communicate regularly with leaders to obtain new member certificates. It may, however, be impractical to let the leader certificates expire because the renewed certificates must be distributed to all the members certified by the recertified leaders.

3.2 Best-effort revocation

The methods described above work well only when time is not critical and communication is guaranteed. Therefore, we have to resort to the distribution of *revocation lists*, which is much faster, albeit less reliable.

In our protocol, a leader may revoke the membership of any members or leaders of the same group. Note that we revoke membership, i.e. $\langle Groupkey, Memberkey \rangle$ pairs, and not keys (as in X.509 or PGP [CCI88,Zim95]) or certificates (as in SPKI). (Leaders are not allowed remove members from other groups, so universal revocation of a key cannot be allowed, and it is more convenient to revoke all the redundant certificate chains at once than to revoke each certificate separately.) The revocation entries have the same structure as the membership certificates. Like the certificates, They must be signed by a leader of the group and they can only be accepted when accompanied by the certificate chain that proves the leadership.

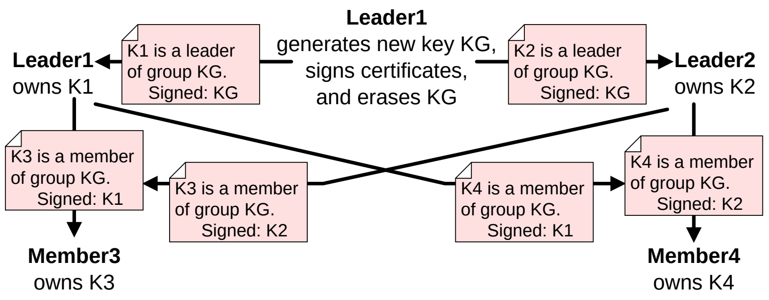
Leader1
generates new key KG,
signs certificates,
and erases KG
K1 is a leader of group KG. Signed: KG
K2 is a leader of group KG. Signed: KG
Leader1 owns K1
Leader2 owns K2
K3 is a member of group KG. Signed: K1
K3 is a member of group KG. Signed: K2
K4 is a member of group KG. Signed: K1
K4 is a member of group KG. Signed: K2
Member3 owns K3
Member4 owns K4

Fig. 3. Redundant certificates and erased group key

Additionally, we allow a key to revoke its own membership in all groups at once. The revocation information may be collected to revocation lists if there is a lot of it. However, group reconstitution and agreed maximum validity times help to reduce the amount of revocation data.

3.3 Minimizing the side effects

A certificate chain is only valid as long as all the certificates in it are. If a leader's membership is revoked, that makes invalid all the certificates signed by that leader. Consequently, all members that depend on such certificate must obtain new ones. Our architecture provides two methods for protecting the members against such effects: redundant certificate chains and erasing the private group key.

The members may obtain multiple certificates from independent leaders. If one chain becomes invalid, another one can still be used to prove the membership. Verification of the certificates is also made faster by using the shortest valid chains.

Redundant certificates will not help if the group key itself is revoked from the group, however. Therefore, it pays to protect the private group key particularly well. The most effective way to prevent the compromise of a private key is, simply, to erase it. After a new group key is generated, it can be used to sign a few leader certificates and then destroyed. The certificates signed with the erased key will remain valid and can still be verified with the public key. That way, it is never necessary to revoke the group key.

In Fig. 3, if either leader loses its authority due to revocation, no other member is affected. Note that the redundant certificates do not complicate the revocation in any way. In fact, the leader revoking a member need not know which certificates have been issued to it. This is why we have chosen to revoke membership and not single certificates.

4 Optimizing revocation

Revocation with unreliable communication is inherently unreliable. Conveying revocation information to all the verifiers may take an arbitrarily long time and some nodes may never be reached. But even when the revocation cannot be guaranteed to succeed, care should be taken to minimize the window of opportunity for the attacker.

We will first describe how revocation information is handled in our architecture and then discuss potential improvements.

4.1 Propagation inside a group

The revocation data needs to be conveyed to all the entities that may verify the membership. These include both outside services and group members.

The primary method of distributing revocation lists in our architecture is propagation from member to member. When two members of the same group come to contact, they should propagate revocation information to each other. This way, the data that one member has eventually spread to all those members that are connected by regular communication.

In order to avoid denial of service by flooding with bogus revocation entries, a node is only required to accept revocation data for the groups in which it knows itself to be a member. This data is easy to recognize because it mentions the group name, it is signed by a leader of the group, and it is accompanied by a certificate chain that proves the authority of that leader. Other data is stored and forwarded only when there is excess capacity.

Clearly, a member should be revoked only when there is a reason to suspect that the private key has been compromised and there is an urgent need to react. Revocation should not be used for scheduled key changes or for replacing accidentally deleted keys. Moreover, fixed maximum validity periods for certificates or regular reconstitution of the group starting from a new group key should be used to keep the revocation lists short.

4.2 Registration of outside services

Since we limit the propagation of revocation information to the group members themselves, some other mechanism must be provided to notify outside services that also may verify group membership. For example, when a group has the right to access a printer, the printer may want to know about removed members.

For this purpose, we borrow a solution from classic revocation literature: the service must register its contact information with some group members, which will then inform the service. These members are often group leaders.

The outside services, like group members themselves, may have limited storage capacity. If a service is not willing to store revocation information and a member of the client group if revoked, the group should unsubscribe the service (have the group key removed from the ACL) until the group has been reconstituted by replacing the group key.

4.3 Subgroups and revocation

When it comes to revocation, we have decided to treat supergroups in the same way as external services. If the supergroup leaders want to receive revocation data from a group, they must register with appropriate members. After receiving a revocation message, the leader may issue a revocation message of the same key also in the supergroup.

This choice is by no means obvious. It means that revocation of a subgroup member will not be passed directly to the supergroup members even if they come to contact with the subgroup before their leaders do. The reason for this is that the supergroup members do not necessarily know about the subgroups and cannot recognize the revocation data as relevant to them. In order to avoid denial-of-service attacks, they cannot be required to store and distribute the data. Non-leader members of the supergroup may also register as recipients of the subgroup revocation data if they know it to be useful.

Information that comes from the supergroup to the subgroup is less problematic. The subgroup members either know about the supergroup or they do not. Only those that are aware or their membership in the supergroup will propagate revocation data for that group. This is natural because the nodes that do not even know about the supergroup certainly do not need to know about revocations.

4.4 Optimal revocation?

While the policies outlined above seem to create a reasonable balance between effectiveness and cost of the revocation for the applications that we have in mind, they are nor guaranteed to be optimal in any respect.

In a theoretical setting, we could aim for information theoretically maximal distribution of revocation lists. We could simply require all nodes to store all revocation data they have ever received and to pass it to all the nodes that they communicate with. In the synchronous model of communication that we have implicitly assumed above, any two or more nodes that establish a communications link should synchronize their revocation lists. In an asynchronous model, the data should be appended to all messages ever sent. Such requirements are clearly unreasonable for real-world systems with limited memory and bandwidth. The threat of denial-of-service attacks also means that the nodes cannot be required to accept any information unless they can verify that it is authentic.

It is an open question how to optimally propagate revocation information in an distributed system where communication between nodes is only occasional and the nodes have capacity limitations. To solve this problem, one should also provide some exact measure for comparing the efficiency of such systems. The theory of gossip-based probabilistic event propagation might provide some tools for the comparison [KMG00]. Since these questions are hard to answer, we have taken a practical approach in this paper and suggested a first suboptimal solution, which hopefully can be improved.

5 Conclusions

We described techniques for secure group membership management in highly distributed systems where nodes are mobile and prone to loss or failure, and communication links between them exist only occasionally. In our architecture, a group can be created ad hoc by generating a new public signature key, which will be used as the group identifier and to certify the public keys of other group leaders and members. The leaders may act independently in certifying further leaders and members, and in revoking them. The optimal propagation of revocation lists with given resources and unreliable communication was identified as an open problem.

Acknowledgments

This research was funded by Defence Forces Research Institute of Technology, Finland, and supported by Helsinki Graduate School in Computer Science and Engineering (HeCSE) and Academy of Finland grant #47754. The work was done mostly while Tuomas Aura was at HUT. The group architecture was originally presented in Nordsec2000 [MAH00].

References

[AG00] N. Asokan and Philip Ginzboorg. Key-argeement in ad-hoc networks. *Elsevier Preprint*, 2000. To appear.

[AST00] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–640, April 2000.

[BD94] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology - EUROCRYPT '94*, volume 950 of *LNCS*, pages 275–286, Perugia, Italy, May 1994. Springer.

[BFIK99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos Keromytis. The KeyNote trust-management system version 2. RFC 2704, IETF Network Working Group, September 1999.

[Blu99] Specification of the Bluetooth system, version 1.0b, 1999.

[CCI88] CCITT. *Recommendation X.509, The Directory - Authentication Framework,* volume VIII of *CCITT Blue Book,* pages 48–81. 1988.

[CJS99] Wenli Chen, Nitin Jain, and Suresh Singh. ANMP: ad hoc network management protocol. *IEEE Journal on Selected Areas in Communication*, 17(8):1506–1531, August 1999.

[Des94] Yvo G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July–August 1994.

[EFL+99a] Carl Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. Simple public key certificate. Internet draft, July 1999. Work in progress.

[EFL+99b] Carl Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. SPKI certificate theory. RFC 2693, IETF Network Working Group, September 1999.

[IEE97] IEEE Standards Board. 802 part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1997.

[KMG00] Anne-Marie Kermarrec, Laurent Massoulie, and Ayalvadi J. Ganesh. Reliable probabilistic communication in large-scale information dissemination systems. Technical Report MMSR-TR-2000-105, Microsoft Research, Cambridge, UK, October 2000.

[MAH00] Silja Mäki, Tuomas Aura, and Maarit Hietalahti. Robust membership management for ad-hoc groups. In *Proc. 5th Nordic Workshop on Secure IT Systems (NORDSEC 2000)*, Reykjavik, Iceland, October 2000.

[RL96] Ronald L. Rivest and Butler Lampson. SDSI - a simple distributed security infrastucture. Technical report, April 1996.

[Roe97] Gert Roelofsen. TETRA security - the fundament of a high performance system. In *Proc. TETRA Conference 1997*, 1997.

[ZH99] Lidong Zhou and Zygmunt J. Haas. Securing ad hoc networks. *IEEE Network Magazine*, 13(6), November-December 1999.

[Zim95] Philip Zimmermann. *The Official PGP User's Guide*. MIT Press, June 1995.