Distributed registration and key distribution (DiRK)

Rolf Oppliger
University of Berne, Institute for Computer Science
Neubrückstrasse 10, CH-3012 Berne, Switzerland
Phone +41 31 631 89 57, Fax +41 31 631 39 65, oppliger@iam.unibe.ch

Andres Albanese
International Computer Science Institute (ICSI)
1947 Center Street, Berkeley, CA 94704-1105, USA
Phone (510) 643-9365, Fax (510) 643-7684, aa@icsi.berkeley.edu

Abstract

This paper proposes a technique that can be used in a conferencing system to handle participants registration and key distribution in a decentralized and distributed way. The technique is named DiRK, an acronym derived from Distributed Registration and Key distribution. The basic idea of DiRK is to distinguish between active and passive participants in a particular conference session, and to have the active participants assist the session holder to register participants and distribute session keys accordingly. As an exemplary application, the paper also addresses the use of DiRK in the context of multimedia conferencing over the Internet Multicast Backbone (MBone).

Keywords

Conferencing system, participants registration, key distribution, IP multicast, MBone

1 INTRODUCTION

In real life, a conference is held if two or more parties want to discuss a specific topic. The conference is private, if participation is restricted, and it is public, if participation is not restricted. A private conference is typically held behind closed doors if the privacy of the discussed topic(s) must be protected, too. Note that a private conference always requires participants registration and authentication, whereas a public conference may require either participants registration, or both participants registration and authentication. For example, a scientific conference may be public by nature, but require participants registration for billing purposes. In addition to that, the conference may require authentication for providing membership discounts. In general, there are four types of conferences to be distinguished:

- Public conferences
- Public conferences with participants registration
- Public conferences with participants registration and authentication
- Private conferences

In a multi-party communication system two or more parties can send and receive data, and communicate accordingly. In this report, the term conferencing system is used to refer to a multi-party communication system that meets some realtime requirements. This definition is fairly broad, and the distinction between a multi-party communication system and a conferencing system is thus not too precise.

This paper proposes a technique that can be used in a conferencing system to support all types of conferences itemized above, and to handle participants registration and key distribution in a decentralized and distributed way. The technique is named DiRK, an acronym derived from Distributed Registration and Key distribution. The rest of the paper proceeds as follows: DiRK is briefly overviewed in section two, and the cryptographic protocols that it is based on are described and further discussed in sections three and four. As an exemplary application, the use of DiRK in the context of multimedia conferencing over the Internet Multicast Backbone (MBone) is addressed in section five, and conclusions are drawn in section six.

2 OVERVIEW

In general, a conferencing system allows several conference sessions to take place simultaneously. Every conference session is initiated by a particular user, namely the session holder. It is up to the session holder to specify the conference session type, and to handle participants registration and key distribution accordingly. In this matter, the session holder can follow either a centralized or a decentralized and distributed approach: In a centralized approach, he handles participants registration and key distribution himself, whereas in a decentralized and distributed approach, he nominates agents to assist him. It is assumed that a decentralized and distributed approach better fits the requirements of a conferencing system, and this assumption has driven the current design of DiRK. The basic idea is to distinguish between active and passive participants in a particular conference session, and to have the active participants assist the session holder to register participants and distribute session keys accordingly. In principle, any user can register as an active participant, and subsequently register new users as (active or passive) participants on the session holder's behalf. In addition to the registration of new users, an active participant can also provide them with a session key that may be needed to participate in a private conference session.

However, in a decentralized and distributed approach the session holder or any other user or session participant may be concerned about how to figure out, if a particular participant is registered for a session, and if he is, who has actually registered him, and provided him with the session key. It is important to note that DiRK has been designed to handle participants registration and key distribution in a decentralized and distributed way without loosing this kind of traceability.

It has already been mentioned that DiRK is based on cryptographic protocols that are described and discussed in the following sections. In general, a protocol specifies the format and relative timing of messages exchanged between communicating parties. A cryptographic protocol is a protocol that uses cryptography, meaning that all or parts of the messages are encrypted on the sender' side and decrypted on the receiver' side. The reader of this paper is assumed to be familiar with cryptology in general, and the use of cryptographic protocols in particular (Schneier, 1994). The following notation is used to describe cryptographic protocols:

- $\mathcal{S}$ is used to refer to a particular conference session. It is assumed that $\mathcal{S}$ overlays a particular (multicast or broadcast) channel, and $\mathcal{S}$ is used to refer to this channel, too.
- Capital letters, such as A, B, C, ..., are used to refer to arbitrary users.
- K is used to refer to a secret key, and (k, k^{-1}) is used to refer to a public key pair. In either case, key subscripts may be used to denote users. Capital letters are used for long-term keys, and small letters are used for short-term keys.
- The term $\{m\}K$ is used to denote a message m that is encrypted with the secret key K. The same key is used for decryption, so $\{\{m\}K\}K$ equals m. Similarly, the term $\{m\}k$ is used to denote a message m that is encrypted with the public key k. The message can be decrypted only with the corresponding private key k^{-1}. In a digital signature scheme, a private key is used to digitally sign messages, and the corresponding public key is used to verify the signatures. In this case, the term $\{m\}k^{-1}$ is used to denote a digital signature giving message recovery, and the term $\langle m\rangle k^{-1}$ is used to denote a digital signature with appendix. Note that in the second case, $\langle m\rangle k^{-1}$ abbreviates $m, \{h(m)\}k^{-1}$, with h being a collision-resistant one-way hash function.
- In accordance with international standardization, the term $X \ll Y \gg$ is used to denote a certificate that has been issued by X for Y's long-term public key k_Y. In general, this certificate may conform to any standard in use today, such as e.g. ITU-T X.509.
- In addition to that, the term $X < Y, \mathcal{S} >$ is used to denote a registration certificate that has been issued by X for Y and session $\mathcal{S}$. In this case, X is necessarily registered as an active participant, whereas Y can be registered either as an active or passive participant. If Y is registered as an active (passive) participant, $X < Y, \mathcal{S} >$ abbreviates $\langle X, Y, \mathcal{S}, k_y\rangle k_x^{-1}$ ($\langle X, Y, \mathcal{S}\rangle k_x^{-1}$).

3 CRYPTOGRAPHIC PROTOCOLS

This section describes the cryptographic protocols that DiRK is based on. In particular, it focuses on the session initialization, participants registration, registration validation, and session rekeying protocols.

3.1 Session Initialization

In order to initialize a conference session $\mathcal{S}$, a user A has to randomly select a public key pair (k_a, k_a^{-1}) for $\mathcal{S}$, and announce $\mathcal{S}$ and k_a in public. More precisely, A has to digitally sign the announcement with his long-term private key k_A^{-1}, and make $\langle \mathcal{S}, k_a\rangle k_A^{-1}$ publicly available, e.g. by publishing it on the Web or by periodically sending it on a multicast or broadcast channel. Anybody who knows the long-term public key k_A of A can verify the authenticity of the conference session announcement, and extract $\mathcal{S}$ and k_a accordingly. Note that k_a^{-1} is kept secret by A. This key represents the master key of the session, and the holder of this key is considered as being the current session holder of $\mathcal{S}$. If $\mathcal{S}$ is going to be a private conference session, A has to additionally select a session key $K_{\mathcal{S}}$ for $\mathcal{S}$.

3.2 Participants Registration

Having initiated the session $\mathcal{S}$ as described above, A is notably the first active participant of $\mathcal{S}$. If another user B wants to register as participant, it has to be distinguished, whether he wants to register as active or passive participant. In the first case, B has to randomly select a short-term public key pair (k_b, k_b^{-1}) for $\mathcal{S}$, whereas in the second case, B doesn't have to do so. In either case, B has to run the following participants registration protocol:

$$\begin{array}{lll} 1: \mathrm{B} & \longrightarrow & \mathcal{S}: \mathrm{REG\_REQ}([k_b]) \\ 2: \mathrm{X} & \longrightarrow & \mathcal{S}: \mathrm{REG\_CONF}(B, X < B, \mathcal{S} >, [\{K_{\mathcal{S}}\}k_b]) \end{array}$$

In step 1, B sends a registration request (REG_REQ) message to $\mathcal{S}$. The message includes k_b if B wants to register as an active participant. In principle, any active participant X can grab the registration request, and return a registration confirmation (REG_CONF) message to $\mathcal{S}$ in step 2. This message includes B and a registration certificate $X < B, \mathcal{S} >$ for B and $\mathcal{S}$. Note that $X < B, \mathcal{S} >$ abbreviates $\langle X, B, \mathcal{S}, k_b \rangle k_x^{-1}$ if B is granted active participation, and $\langle X, B, \mathcal{S} \rangle k_x^{-1}$ if B is granted passive participation. If $\mathcal{S}$ is a private conference session, then the registration confirmation additionally includes $\{K_{\mathcal{S}}\}k_b$, which is the session key $K_{\mathcal{S}}$ encrypted with B's public key k_b. Only B is assumed to hold the private key k_b^{-1} that is needed to decrypt $\{K_{\mathcal{S}}\}k_b$, and extract $K_{\mathcal{S}}$ accordingly.

It is characteristic for DiRK that the registration request and the registration confirmation are both sent to $\mathcal{S}$, where they can be grabbed by X and B respectively. The main advantage of having both messages sent to the channel that is associated with the session is that it doesn't require B and X to communicate directly with each other. However, several active participants may simultaneously return a registration confirmation for B in step 2. In order to avoid generating a rapid burst of traffic for every registration request, timers can be used to randomly delay the time before a registration confirmation is actually sent. In addition to that, if an active participant recognizes that another participant has already sent a registration confirmation, he does not send his confirmation either.

The following slight modification of the REG_REQ message is necessary for the participants registration protocol to provide authentication:

$$1: \mathrm{B} \quad \longrightarrow \quad \mathcal{S}: \mathrm{REG\_REQ}(\langle [k_b], Y \ll B \gg \rangle k_B^{-1})$$

In this case, the REG_REQ message carries a certificate $Y \ll B \gg$ for B's long-term public key k_B, and a digital signature that B generates with the corresponding long-term private key k_B^{-1}. Again, the message includes k_b if B wants to register as an active participant. Any active participant X who is willing and able to verify $Y \ll B \gg$ can grab the registration request, extract the public key k_B from $Y \ll B \gg$, verify the appended digital signature with this key, and return a registration confirmation for B in step 2. The corresponding REG_CONF message is similar to the one sent in the participants registration protocol without authentication, and thus omitted in the protocol description above. It has already been mentioned that $Y \ll B \gg$ can conform to any standard in use today. Either the session holder specifies the valid certificates for $\mathcal{S}$, or every participant individually decides what certificates he's going to accept.

3.3 Registration Validation

In most conferencing systems, the participants of a particular conference session periodically send a status report to the channel that is associated with the session. The aim of this status report is to inform the other participants about the current participation. The basic idea of DiRK is to couple this status report with a message that allows other participants to validate the registration of the sending participant. If the sending participant is a passive participant, the coupled message is just the registration certificate. Otherwise, if the sending participant is an active participant, the coupled message consists of both the registration certificate and a timestamp that is digitally signed with the private key that corresponds to the public key that is included in the registration certificate. In either case, the registration validation protocol consists of a single message that participant B periodically sends to $\mathcal{S}$:

$$\mathrm{B} \quad \longrightarrow \quad \mathcal{S} : \mathrm{REG\_VAL}(B, X < B, \mathcal{S} >, [\{T\}k_b^{-1}])$$

The registration validation (REG_VAL) message includes B and $X < B, \mathcal{S} >$, which is the registration certificate that B has received during his registration. If B happens to be an active participant, then the message additionally includes $\{T\}k_b^{-1}$, which is a timestamp T digitally signed with B's private key k_b^{-1}. Any user C can grab the REG_VAL message that B sends to $\mathcal{S}$, and validate the registration of B accordingly:

- If B happens to be a passive participant, then the registration validation message consists of B and $X < B, \mathcal{S} >$, with $X < B, \mathcal{S} >$ abbreviating $\langle X, B, \mathcal{S} \rangle k_x^{-1}$. In this case, B's registration certificate can only be validated with the certificate issuer X's public key k_x. The registration validation of B thus requires the registration validation of X, and the extraction of k_x from X's REG_VAL message. In general, the registration validation of a passive participant requires both the validation of the active participant that has registered him, and the verification of the digital signature that is appended to the registration certificate.
- If B happens to be an active participant, then the registration validation message consists of $B, X < B, \mathcal{S} >$, and $\{T\}k_b^{-1}$, with $X < B, \mathcal{S} >$ abbreviating $\langle X, B, \mathcal{S}, k_b \rangle k_x^{-1}$ and $\{T\}k_b^{-1}$ being a timestamp T digitally signed with k_b^{-1}. In this case, C can extract k_b from B's registration certificate, decrypt T with k_b, and compare the result with a timestamp T' that he takes from his local clock. If the difference between T and T' is less or equal than a certain threshold value, C assumes the REG_VAL message to be fresh, and the registration of B to be valid.

Note that if C simply extracts k_b from B's registration certificate $X < B, \mathcal{S} >$, he can't be sure that the key he extracts is indeed B's key. To make sure that the key is authentic, C has to verify the digital signature that is associated with $X < B, \mathcal{S} >$, too. But in order to verify this signature C must have X's public key k_x. C can extract k_x from X's REG_VAL message, but again, in order to make sure that this key is authentic, C must verify the digital signature that is associated with this message, too. This continues, until C reaches a public key that is certified with A's private key k_a^{-1}. The corresponding public key k_a has been made publicly available as part of the session initialization, and is thus assumed to be authentic.

3.4 Session Rekeying

The legitimate participants of a private conference session can extract the session key from their registration confirmation. However, it is usually good practice to periodically change a session key. DiRK provides three session rekey protocols.

Full Session Rekey

If an active participant A wants to fully rekey the session $\mathcal{S}$, he randomly selects a new session key $K'_{\mathcal{S}}$, and runs the following full session rekey (FSR) protocol:

$$\text{A} \quad \longrightarrow \quad \mathcal{S} : \text{FSR}(\langle\{T, K'_{\mathcal{S}}\}K_{\mathcal{S}}\rangle k_a^{-1})$$

The protocol consists of a single FSR message that A sends to $\mathcal{S}$. The message includes T and $K'_{\mathcal{S}}$, both encrypted with the session key $K_{\mathcal{S}}$, and digitally signed with A's private key k_a^{-1}. T specifies the time to execute the key change and $K'_{\mathcal{S}}$ is the new session key. Note that, in principle, every user knows k_a and can verify the digital signature that is appended to the message, whereas only the participants of $\mathcal{S}$ know $K_{\mathcal{S}}$ and can decrypt $K'_{\mathcal{S}}$ accordingly. Also note that the FSR protocol doesn't allow participants to be excluded from receiving the new session key $K'_{\mathcal{S}}$, and that a selective session rekey protocol is required in this case.

Selective Session Rekey

If A wants to selectively rekey m participants of the session $\mathcal{S}$, he randomly selects a new session key $K'_{\mathcal{S}}$, and runs the following selective session rekey (SSR) protocol:

$$\text{A} \quad \longrightarrow \quad \mathcal{S} : \text{SSR}(\langle T, B_1, \{K'_{\mathcal{S}}\}k_{b_1}, B_2, \{K'_{\mathcal{S}}\}k_{b_2}, \ldots, B_m\{K'_{\mathcal{S}}\}k_{b_m}\rangle k_a^{-1})$$

Again, the protocol consists of a single SSR message that A sends to $\mathcal{S}$. The message includes a timestamp T that specifies the time to execute the key change, and a set of pairs $(B_i, \{K'_{\mathcal{S}}\}k_{b_i})$, each providing a participant with the new session key $(i = 1, \ldots, m)$. As a matter of fact, participant B_i can decrypt $\{K'_{\mathcal{S}}\}k_{b_i}$ with his private key $k_{b_i}^{-1}$. Again, the message is digitally signed with A's private key k_a^{-1}.

Distributed Session Rekey

If A wants to selectively rekey the session $\mathcal{S}$ in a distributed way, he randomly selects a new session key $K'_{\mathcal{S}}$, and runs the following distributed session rekey (DSR) protocol:

$$
\begin{array}{lcl}
1 : \mathrm{A} & \longrightarrow & \mathcal{S} : \mathrm{DSR}(\langle\{T, N, PRL\}K_{\mathcal{S}}\rangle k_a^{-1}) \\
2 : \mathrm{B} & \longrightarrow & \mathcal{S} : \mathrm{DSR\_REQ}(\{\{N\}k_b^{-1}\}K_{\mathcal{S}}) \\
3 : \mathrm{X} & \longrightarrow & \mathcal{S} : \mathrm{DSR\_CONF}(\{B, N, \{K'_{\mathcal{S}}\}k_b\}K_{\mathcal{S}})
\end{array}
$$

In step 1, A sends a DSR message to $\mathcal{S}$. The message includes a timestamp T that specifies the time to execute the key change, a nonce N, and a participation revocation list (PRL). The PRL enumerates the participants that are not allowed to receive the new session key. The message is both encrypted with the old session key $K_{\mathcal{S}}$, and digitally signed with A's private key k_a^{-1}. After having received the DSR message, it is up to participant B to send a distributed session rekey request (DSR_REQ) message to $\mathcal{S}$ in step 2. The message includes $\{\{N\}k_b^{-1}\}K_{\mathcal{S}}$, which is the nonce N digitally signed with B's private key k_b^{-1} and encrypted with $K_{\mathcal{S}}$. The message can be grabbed by any participant X who is already equipped with $K'_{\mathcal{S}}$. X validates the registration of B and makes sure that B is not enumerated in the PRL, before he returns a distributed session rekey confirmation (DSR_CONF) message to $\mathcal{S}$ in step 3. The message includes B, N, and $\{K'_{\mathcal{S}}\}k_b$, which is the new session key $K'_{\mathcal{S}}$ encrypted with B's public key k_b. In addition to that, the message is also encrypted with $K_{\mathcal{S}}$. Consequently, B must have both $K_{\mathcal{S}}$ and k_b^{-1} to successfully decrypt and extract $K'_{\mathcal{S}}$.

Note that the same mechanisms that avoid generating a rapid burst of traffic in the participants registration protocol can be used in the DSR protocol, too. Also note that all session rekey protocols have been described in a way that A is an arbitrary active participant of session $\mathcal{S}$. In practice, it might be more appropriate to set up a security policy that is more restrictive, and to allow only the session holder to actually rekey a session. In this case, the private key k_a^{-1} that is used to digitally sign the initiating session rekey messages must be the master key of the session.

4 DISCUSSION

The working method of DiRK can be best illustrated by using a participants registration tree (PRT). A PRT is a session-specific data structure. In other words, every conference session $\mathcal{S}$ within a conferencing system has a PRT of its own, and this PRT is further referred to as $PRT(\mathcal{S})$. The nodes of $PRT(\mathcal{S})$ correspond to the (active and passive) participants of $\mathcal{S}$, and the edges represent the registration process. In particular, an edge leading from A to B implies that A has registered B for $\mathcal{S}$. As a consequence of the fact that only active participants are allowed to register users as session participants, active participants can be found anywhere in a PRT, whereas passive participants can only be found in the leaves of the tree. A PRT is a dynamic data structure that may change over time. Note that the PRT is stored in a fully distributed way. In fact, every participant needs to store only the edge that leads to his node. However, it might be useful for backup purposes to have every node store the edges that lead down the PRT, too. Also note that a PRT, in general, doesn't imply any temporal ordering within the registration process.

When a user A initializes a conference session $\mathcal{S}$, he creates the root of the $PRT(\mathcal{S})$. An edge is added, whenever A registers a new user as a session participant of $\mathcal{S}$. The edge necessarily leads down to a leave in the tree, if the new user is registered as a passive participant. Otherwise the new user is registered as an active participant that can afterwards register new users as (active or passive) participants, too.

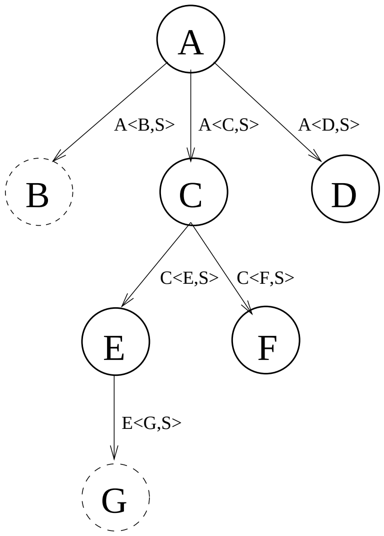

Figure 1 Participants registration tree (PRT).

Figure 1 illustrates a PRT for a conference session with seven participants (A, B, C, D, E, F, and G). The active participants (A, C, D, E, and F) are represented in solid circles, whereas the passive participants (B and G) are represented in dotted circles. The figure can be read as follows: The session is initiated by user A who represents the root of the PRT. A has registered B as passive participant, and both C and D as active participants. C has registered E and F as active participants, and E has finally registered G as passive participant. Note that if a user H wanted to register for $\mathcal{S}$, he would send a registration request that, in principle, could be confirmed by any of the active participants.

If D wanted to validate the registration of G, he would have to extract k_g from $E < G, \mathcal{S} >$, which is part of G's REG_VAL message, decrypt the timestamp T with k_g, and compare the result with his local time. If the difference were less or equal than a certain threshold value, he would accept the registration of G to be valid. But in order to extract k_g from $E < G, \mathcal{S} >$, D has to get the public key of E. He therefore has to grab E's and C's REG_VAL messages and extract k_e and k_c from $C < E, \mathcal{S} >$ and $A < C, \mathcal{S} >$ accordingly. Equipped with these keys, D is able to validate the registration of G. Note that, in principle, the registration validation of G requires the registration validation of every participant that is on the path from A down to G in the corresponding PRT. This path is independent from who is actually validating the registration of G. It is even possible to set up a registration validation service in a conferencing system that periodically informs about the current participation of conference sessions.

5 APPLICATION

As an exemplary application, this section addresses the use of DiRK in the context of multimedia conferencing over the Internet Multicast Backbone (MBone). The MBone is an experimental network that is overlaid on the existing Internet to carry IP multicast datagrams (Deering and Cheriton, 1990). Application tools, such as `vat` (visual audio tool), `nevot` (network voice terminal), `nv` (network video), `ivs` (inria videoconferencing system), `vic` (video conferencing) and `wb` (whiteboard) are used to hold multimedia conference sessions over the MBone. In addition to that, the `sd` (session directory) tool can be used to announce MBone conference sessions in public, and to have MBone application tools start up accordingly (Macedonia and Brutzman, 1994).

In spite of the various threats that are related to IP multicast, security-related issues for the MBone have rarely been touched upon to date (Ballardie and Crowcroft, 1995). As a matter of fact, IP multicast provides the users with little control over who participates in a conference session, the extent of participation of each user, or to which users the messages are delivered. There is, in general, no control as to who can register a dynamic session address, who can or cannot join a session, who can receive traffic from or inject traffic into the session, or how to deal with the security implications when a host leaves a session. Consequently, MBone application tools only support public conferencing, and an immature form of private conferencing:

- With regard to public conferencing, there is currently no possibility to register users as participants of particular MBone conference sessions, and to authenticate them accordingly.
- With regard to private conferencing, some MBone application tools support session encryption. The idea is to encrypt (decrypt) every datagram that is sent to (received from) an IP multicast address, and to have the participants share a session key. However, the MBone application tools that support session encryption currently don't address key distribution either, and relay on out-of-band key distribution mechanisms instead.

In order to fully support private conferencing over the MBone, key distribution facilities must be added to the corresponding application tools. One possibility is to enhance the `sd` tool with key registration facilities, and to use secure e-mail systems, such as Privacy Enhanced Mail (PEM) or Pretty Good Privacy (PGP), to distribute session keys. In this case, the organizer or holder of a MBone conference session would randomly select a session key, forward it to the session participants, and encrypt the session announcement with this key. Whenever the `sd` tool of a MBone user received an encrypted session announcement, it would try to decrypt it with all registered keys. If it sucessfully decrypted the announcement, it would display the announcement and remember the key for that particular session. When the session started, the `sd` tool would hand over the session key to the other MBone application tools. This approach is simple and straightforward. It is being followed by the Lawrence Berkeley Laboratory (LBL) and the European MICE (Multimedia Integrated Conferencing for European researchers) project. However, it is assumed that the approach is too centralized to be used on a large scale. In addition to that, the approach does not address participants registration and authentication either.

With regard to future MBone application tools, DiRK can be used to handle participants registration and key distribution in a decentralized and distributed way. Compared to any centralized approach, DiRK has scalability advantages and may work more efficiently. In principle, DiRK can be used either build a session registration tool (`srt`) for the MBone, or to implement a toolbox to be incorporated into other MBone application tools. In the first case, the `srt` would allow users to create, edit, and possibly announce MBone conference sessions, to register as participants, and to validate the registration of others. Similar to the `sd` tool, the `srt` would overlay a specific IP multicast address. In the second case, the toolbox would provide an application programming interface (API) that could be used to incorporate DiRK functionality into other MBone application tools.

6 CONCLUSIONS

This paper has proposed a technique that can be used in a conferencing system to handle participants registration and key distribution in a decentralized and distributed way. The technique is named DiRK, an acronym derived from Distributed Registration and Key distribution. As an exemplary application, the paper has also addressed the use of DiRK in the context of multimedia conferencing over the Internet Multicast Backbone (MBone). In this context, DiRK can be used either to build a session registration tool (`srt`) for the MBone, or to implement a toolbox that can be incorporated into other MBone application tools to provide DiRK functionality. In a first step, a `srt` is being prototyped at the University of Berne. This prototype implementation will serve as a starting-point to further investigate on the requirements of participants registration and key distribution in multicast environments. Recent studies have shown that DiRK meets the requirements of online universities. Further information on DiRK and its use in online universities can be found on the Web by following the URL `http://iamwww.unibe.ch/~oppliger/dirk.html`.

REFERENCES

Ballardie, T. and Crowcroft, J. (1995) Multicast-Specific Security Threats and Counter-Measures. Internet Society Symposium on Network and Distributed System Security, 2-16.

Deering, S. and Cheriton, D.R. (1990) Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2), 85-110.

Macedonia, M.R. and Brutzman, D.P. (1994) MBone Provides Audio and Video Across the Internet. *IEEE Computer*, 27(4), 30-36.

Schneier, B. (1994) *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, New York.

BIOGRAPHY

Rolf Oppliger studied computer science, mathematics, and economics at the University of Berne, Switzerland, where he received his M.Sc. and Ph.D. degrees from the Institute for Computer Science and Applied Mathematics (IAM). His research interests are directed towards cryptography, and the use of cryptographic techniques in computer networks and distributed systems. He's currently with the Swiss Federal Office of Information Technology and Systems (BFI) and the University of Berne. He also serves as a vice-chair of IFIP TC 11/WG 4 on network security.

Andres Albanese holds a Ph.D. degree in electrical engineering from Stanford University, a M.Sc. in the same field from the University of Texas at Austin, and an “Ingeniero Electricista” degree from Universidad Central de Venezuela. He has been working in the areas of fiber optics, broadband communications, and high-speed computer networks for several years. His present research interest are in group multimedia applications for global networks. He has published widely and holds 13 patents. He's currently vice president at the International Computer Science Institute (ICSI) in Berkeley, California.