The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution

Ray Bird, Inder Gopal, *Fellow, IEEE,* Amir Herzberg, Phil Janson, Shay Kutten, *Member, IEEE,* Refik Molva, *Member, IEEE,* and Moti Yung

***Abstract*— An essential function for achieving security in computer networks is reliable authentication of communicating parties and network components. Such authentication typically relies on exchanges of cryptographic messages between the involved parties, which in turn implies that these parties be able to acquire shared secret keys or certified public keys. Provision of authentication and key distribution functions in the primitive and resource-constrained environments of low-function networking mechanisms, portable, or wireless devices presents challenges in terms of resource usage, system management, ease of use, efficiency, and flexibility that are beyond the capabilities of previous designs such as Kerberos or X.509.**

This paper presents a family of light-weight authentication and key distribution protocols suitable for use in the low layers of network architectures. All the protocols are built around a common two-way authentication protocol. The paper argues that key distribution may require substantially different approaches in different network environments and shows that the proposed family of protocols offers a flexible palette of compatible solutions addressing many different networking scenarios. The mechanisms are minimal in cryptographic processing and message size, yet they are strong enough to meet the needs of secure key distribution for network entity authentication.

The protocols presented have been implemented as part of a comprehensive security subsystem prototype called KryptoKnight, whose software and implementation aspects are discussed in [16], and which is the basis for the recently announced IBM Network Security Program product.

I. INTRODUCTION

THE STEADILY growing use of computer networks is fostering increasing concerns about network security. One of the main issues is effective control of access to network components and resources by expanding populations of users, some of which cannot always be trusted to use the network properly. An essential requirement for implementing suitable access controls is a mechanism for reliable authentication of communicating entities and network users.

State-of-the-art authentication protocols typically rely on exchanges of cryptographic messages between entities wanting to authenticate one another prior to or during actual communication. Such exchanges in turn demand that the communicating parties be able to acquire cryptographic keys for performing the necessary authentication exchanges—these keys may be secret keys shared by the involved parties if symmetric cryptography is used, or public keys certified by a trusted authority if asymmetric cryptography is used. The issues of authentication and key distribution in the context of distributed applications have already been largely addressed by designs such as Kerberos [24] and X.509 [12].

However, low-function networking mechanisms are typically *subject to various environmental restrictions* (e.g., small packet header sizes, limited computing resources), which must be taken into account when integrating security features. Such mechanisms must also accommodate various network configurations and connectivities (e.g., local area networks, wireless networks, global networks, etc.) and devices of widely differing capabilities (e.g., palm-top, lap-top, desk-top, or floor-standing devices). In such environments, the most primitive component must be able to inter-operate securely with the most powerful one, which implies the provision of simple common security functions to meet minimal needs. Getting essential authentication and key distribution functions to work within the resource restrictions of such low-function networking environments presents challenges in terms of resource usage, efficiency, ease of use, flexibility, and system management. These issues are the subject of the present paper.

The paper explains why existing key distribution designs, aimed at distributed application environments, are not suitable for use in low-function devices and network protocols. Instead the paper proposes a family of light-weight, authenticated key distribution protocols that are especially suited for use in such restricted environments. (The proposed protocols could also be used at the application layer, although their intrinsic economy may lack some of the functional features often desired in distributed applications.) The paper finally argues that key distribution may require different approaches (e.g., communication patterns) in different network environments and it shows that the proposed family of protocols offers a flexible palette of compatible solutions addressing many possible cases.

II. BACKGROUND AND RELATED WORK

A. *Key Distribution—Existing Designs and Limitations*

In a large network with tens, hundreds, or thousands of switching nodes and server computers, and several orders of magnitude more client workstations and users, the problem of the distribution and safe-keeping of cryptographic keys is not

Manuscript received December 7, 1992; revised May 18, 1993 and June 30, 1993; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Lam.

R. Bird is with IBM Corporation, Research Triangle Park, NC 27709 USA.
I. Gopal, A. Herzberg, S. Kutten, and M. Yung are with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA.
P. Janson is with IBM Zurich Research Laboratory, 8803 Ruschlikon, Switzerland (e-mail: pj@zurich.ibm.com).
R. Molva is with Institut Eurecom, 06903 Sophia-Antipolis, France.

IEEE Log Number 9408394.

trivial. It is complex enough that one must abandon right away the idea of casual (manual) distribution procedures outside the system. Automated techniques based on electronic exchanges over the network are required.

Several schemes have been designed for automating key distribution services in distributed systems, e.g., [18], [7], [1], [19], [21], [14], [20]. Some of these have lead to notable implementations or standards, e.g., the Kerberos [24] and ANSI X9.17 [11] schemes, which use DES technology, or the ISO-CCITT X.509 Directory Authentication scheme [12], which relies on public key technology. These schemes vary in popularity. Kerberos is the basis for the security component provided by the Open Software Foundation (OSF^{TM}) as part of its Distributed Computing Environment (DCE) product. However the authentication and key distribution protocols defined by these designs are targeted at distributed application environments, and thus present a number of features that make their use cumbersome or undesirable in more primitive environments such as low-function link, network, or transport protocols, or remote booting protocols.

Yet authentication and key distribution functions are also required within lower-function layers. For instance, most wireless link protocols (e.g., CDPD, GSM) rely on such functions to assert the identity of a calling mobile device (though key distribution designs are rather primitive and not very secure); similar functions can and have been built into X.25 packet layer implementations to restrict incoming calls to certain identified calling addresses; LAN adapter-based booting protocols would also need such functions to prevent booting from untrusted boot servers. Thus, in contrast to the existing application-layer designs, our goal here is to design light-weight key distribution protocols that require minimal computing resources from their environment. A common authentication protocol that has been tested against a wide range of attacks, called interleaving attacks, serves as the foundation for the entire family of key distribution protocols. It is described in the following section.

B. Two-Way Authentication—Basic Building Block

As suggested in the introduction, one of the most basic functions required to enforce any security in a network is the ability for communicating parties to prove their identities to one another. State-of-the-art techniques for providing such authentication rely on exchanges of cryptographic messages. Many such protocols have been designed, some have been standardized, e.g., X.509 [12], or are in common use, e.g., as part of the Kerberos system [24]. In all these designs, each party proves its identity to the other by demonstrating knowledge of a secret, typically a cryptographic key, through encryption of a challenge (time-stamp, counter, or one-time random number) that is unique in that it changes with every execution of the protocol. However many of these designs present a number of features that make their use cumbersome or undesirable in the primitive environments of link, network, or transport protocols below the application layer. They typically require exchanges of unnecessarily large cryptographic messages and involve relatively heavy computations,

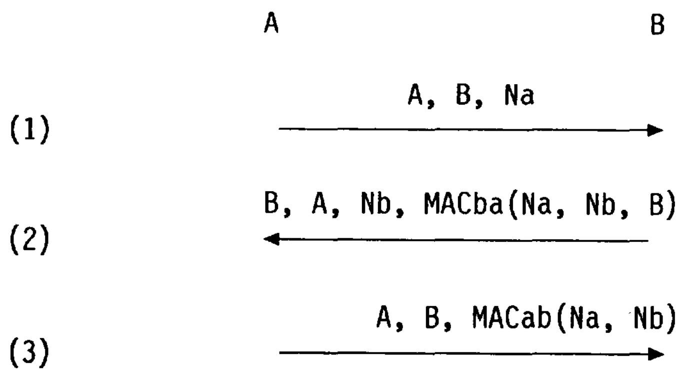


Fig. 1. Secure two-way authentication protocol.

especially where public-key cryptography is used. Simpler protocols can and indeed have been designed. However, while simplicity may give the illusion of security, it is in fact hard to achieve. In [5], [6], we introduced a family of authentication protocols that are at the same time efficient in message size and computation overhead, and resistant to a wide set of attacks known as interleaving attacks. The basic idea for formally proving the correctness of these and similar protocols was discussed initially in [5], while extension and full exploitation of the concept were developed in [9] and [2], after the work reported here was completed. The formal proof concludes that a successful attack would require directly breaking the underlying encryption mechanism (which is assumed to be impractical). This proof is not discussed further in the current paper.

One of the protocols derived from that work is represented in Fig. 1. In this figure, the letters A and B stand for the identifiers of two network entities performing two-way authentication. The variables Na and Nb are one-time random numbers (called **nonces**) used by each party to challenge the other to prove its identity. The notations MACba and MACab denote cryptographic one-way hash functions, called Message Authentication Codes, computed with keys Kba and Kab respectively, and used to guarantee the authenticity of their parameter string. (The commas between the MAC parameters indicate concatenation into one string to which the MAC function is then applied.) Implementing the MAC functions with a symmetric cryptographic system such as DES [8], Kab and Kba are (typically equal) secret keys shared by A and B. Instead of using an encryption function to implement the MAC's, one can use a plain one-way hash function (e.g., the MD5 algorithm [23] is believed to be such a function) applied to the authenticated string concatenated with the secret key [25]. Using such a plain one-way hash function instead of an encryption function has the added advantage that even the source code or the detailed design of the resulting implementation are not subject to the export controls applicable to cryptographic technology in many countries.

In addition to being exportable and secure, the above protocol presents an interesting aspect for low-function networking environments: it requires a minimal amount of computation and minimal-size messages. These features make it particularly well-adapted to securing such primitive network functions as remote booting or link-layer authentication, that are executed in environments where processing power, memory space, and message sizes are limited. Indeed, the longest message (on flow (2)) requires only four variables: the cleartext identifiers A and B of the involved parties (which would already be included in a typical packet header anyway), the cleartext nonce Nb, and the result of the cryptographic MAC computation, which could be as short as 128, 64, or even 32 b, depending on the function used for the MAC computation. Using asymmetric cryptography is of course also possible, but would require much longer messages, and is thus not considered further in this paper. By contrast, in designs such as Kerberos or X.509, initial authentication messages, even leaving out their key distribution aspects, tend to involve more parameters, more redundancy, more cryptography, sometimes also more function to be fair, but function that is not relevant in lower-level network mechanisms. The protocol presented above is the foundation for the family of key distribution designs to be discussed in this paper.

III. Design Issues

We now review the design issues that were considered in the development of secure key distribution mechanisms for primitive devices and low-function resource-constrained networking environments.

A. Restricted Versus Generous Use of Computing Resources

Typical network environments of the future will include WAN stations, LAN stations, wireless stations, portable stations, simple switches, etc., with limited computing resources. Any authentication or key distribution mechanism within such low-function devices must take these resource restrictions into account.

There exist many key distribution designs but most are aimed primarily at the application layer where they can assume a relatively comfortable execution environment and make fairly generous use of computing resources available at that layer. Yet, while key distribution messages can always be generated by key management software executing at high levels in systems with ample resources, they sometimes need to be received and processed by low-function devices with limited resources. For such devices many existing designs, with their comfortable assumptions about the run-time environment, are ill-suited.

The key distribution protocols discussed hereafter were designed with restricted environments in mind. They make minimal use of cryptographic functions, rely on the most basic block-cipher or one-way hash primitives, require little working space and computing power, and limit message size and redundancy as much as possible. In particular, they re-use and avoid repeating clear- or ciphertext parameters that are already present in typical packet headers anyway (e.g., party identifiers) or can be derived from the execution context (e.g., lifetime of a key equal to lifetime of a connection context).

B. Exposure to Export Restrictions

An unfortunate and sometimes surprising consequence of using cryptographic messages, especially large ones, in authentication and key distribution implementations is that they become subject to various administrative controls in countries that have restrictions on the export, import, manufacture, sale, and sometimes even usage of cryptographic devices. However, in many countries restrictions apply to the use of such devices for confidentiality purposes, i.e., when they are used or can easily be used for bulk encryption of secret data. When the devices are designed (tailored) so that they can be used only for computing and verifying the integrity of messages through short (typically less than 128 b) MAC's, using one-way hash functions rather than encryption and decryption functions, their distribution and use poses no problem because one-way functions compress data in a way that does not allow any decompression. Note that applying a one-way hash function to a string concatenated to a secret key may be viewed as generating a pseudo-random function of the string and thus used for encryption. However this is a very restricted, inefficient, and low-bandwidth form of encryption (in contrast to the use of true optimized block encryption algorithms, which are highly suitable for bulk encryption at reasonable speeds).

Like the previously discussed authentication protocol, the key distribution protocols discussed hereafter avoid usage and deployment restrictions by relying only on MAC computations instead of encryption and decryption functions as do the Kerberos, X9.17 and X.509 designs, for instance. Of course secretly transmitting cryptographic keys across networks requires some confidentiality. However, such small secrets can always be hidden using a technique called one-time padding, which involves masking them through exclusive-or operations with one-time random numbers. When restricted to such short secrets as cryptographic keys, this elementary form of encryption is not subject to legal deployment restrictions[1]. The need to generate a random pad, use it for encryption, and exchange a seed for it, does not enable reasonable encryption of large bulk data, and this justifies the fact that there are no export regulations applied to this technique, as explained in the previous paragraph. This padding technique is used in all key distribution protocols proposed hereafter. Designs such as Kerberos, X9.17, or X.509 do not use it.

The usability of global networks connecting numerous users across many businesses requires that authentication services be unrestricted between countries. In a mobile environment where users may move from country to country, the situation would become intractable if authentication protocols were restricted: how could one deal with cases where a user from a financial industry from country A visits country B and communicates with a commercial customer in country C, using equipment produced in country D? Getting a universally accepted technology of which usage is not limited by any government is valuable to side-step the above problems. Such a technology is also sufficient for key distribution purposes.

C. Assumptions on Operational Network Aspects

Most of the existing key distribution methods have been tailored to the relatively narrow operational scenarios of specific

[1] A commercial product derived from this work and using this form of encryption is being market around the world without restrictions.

distributed applications. However such operational assumptions do not necessarily hold in all networking scenarios, and may not even be suitable at all in certain cases.

For instance, any scheme relying on time-stamps favors local area network (LAN) environments, where all network users have easy access to a common trusted time server. While requiring tightly synchronized clocks in a wide area network (WAN) is conceivable, it certainly confers another dimension to the challenge, especially if validating a distributed key requires maintaining a real-time clock by such primitive network components as link adapters, for instance, or agreeing on a standard time zone, or trusting that clocks at other domains are well administered, etc. Any time synchronization (software or hardware) would have to be secure, otherwise an adversary could attack the key distribution protocol by attacking the time synchronization module.

More importantly, existing schemes make specific assumptions about network configuration and connectivity models. They typically assume that keys can be acquired either from a directory (e.g., certified public keys from the X.500 directory) or from some key distribution center (KDC) or authentication server (e.g., X9.17 or Kerberos). Either assumption requires that the entities seeking to acquire keys know how to reach the necessary directory or KDC and have connectivity to it. Existing designs typically dictate a specific communication paradigm for contacting the directory or KDC. For instance, when a client A needs a key to communicate with a server B, Kerberos specifies that A must obtain the desired key from the KDC prior to communicating with B, which is sometimes called the **push model**. By contrast, in the same situation, X9.17 specifies that A must contact B first, and let B get the necessary key from the KDC, which is sometimes referred to as the **pull model**.

Both models are justified in their respective environments. In the LAN environments for which Kerberos was initially designed, requiring the clients rather than the servers to get the keys makes sense because it distributes the burden of getting keys over individual clients, thus alleviating the task of shared servers. In the WAN environments for which X9.17 was designed, the opposite approach is justified because there are typically many more clients than servers, so that the amount of system definition that would be required could be prohibitive. Similarly in mobile networks where a client tries to contact a home server, the KDC is likely to be much closer to that server than to the roaming client, thus saving the expense of long-range client-KDC communication that Kerberos would require.

Dictating a fixed connectivity pattern is possible in well-defined application environments where operating assumptions are clear. However this is unreasonable in generic network environments, where the context may change from time to time and from place to place, and some situations may rule out certain possibilities.

Consider for example a situation where a mobile or otherwise unidentified station dials into a network, and must be authenticated by the nearest switching node (intermediate system) before being allowed to contact any station across the network. This scenario clearly rules out the Kerberos-like push model, where the newcomer should contact the KDC before talking to the switching node. By contrast, consider a situation where two switching nodes of an accidentally partitioned network re-establish contact after the outage and need to authenticate one another. In this case, which of the two nodes should contact the KDC may depend on which network partition the KDC is in. Even in cases where physical connectivity exists, it may be impossible (for route configuration reasons) or undesirable (for security reasons) to provide some logical connectivity. The family of protocols described hereafter offers the flexibility of negotiating connectivity patterns at run-time to communicate with a KDC or directory.

IV. Key Distribution for Network Environments

Used with MD5 hashing, CBC DES MAC, or any other one-way function using a shared key as one of its arguments, the authentication protocol described earlier presents the advantages of requiring only relatively simple and short messages. In addition, with CBC DES MAC or other encryption functions used as one-way functions, the object code of an implementation may be exported as long as it does not offer any handle for bulk encryption/decryption. With MD5, even the source code of an implementation may be exported since it does not embody any reversible encryption function, such as DES for instance. However, MD5 (or DES) requires that pairwise shared secret keys be distributed to any two parties wishing to authenticate one another. This section will describe a suitable key distribution protocol which builds on the basic two-way authentication protocol described earlier, thus capitalizing on its light-weight and exportability properties. In addition, it is flexible and nonprescriptive as far as network configuration and connectivity are concerned, thus supporting both the push and pull models of key distribution.

A. Certificates and Tickets

The distribution of secret keys from a KDC to parties in its network requires that secret channels be used. In a symmetric cryptography context, it is natural to envision securing such channels through symmetric encryption. This in turn requires that each party in the network share with the KDC at least one secret key, which it can use to secure its channel with the KDC. We call this secret key shared between the KDC and one of its clients the client's main **key-encryption key (KEK)**.

The nature of the messages exchanged between a KDC and its clients over the secret channel they share is actually reminiscent of public key certificates: a key distribution message issued by the KDC to one of its clients, also referred to as a **ticket**, must by definition include the distributed key; it must also include a **key stamp** (e.g., a unique key version number, time-stamp, or nonce) indicating to the recipient that the key is fresh and valid; it must finally include the identifier of the party(ies) with which the key is intended to be used. (If it did not include that identifier, an intruder X could substitute one certificate for another, thus tricking a party A into accepting as the key meant to authenticate B some other key that X happens to know, which in turn would allow X to impersonate B.) Thus, secret key tickets contain essentially the same sort of information as public key certificates, the main difference being the size of the distributed key (and, of course, the cryptographic method by which it is authenticated and kept secret as necessary).

In the ensuing discussion, we adopt the following notation:

Kx	Party X's KEK
Kxy	Secret key shared by X and Y
Ex(M)	Encryption of M under key Kx
Exy(M)	Encryption of M under key Kxy
MACx(M)	Message Authentication Code for message M under key Kx
MACxy(M)	Message Authentication Code for message M under key Kxy.

Using this notation, a ticket for giving A a shared secret key Kab to communicate with B can, assuming E also provides integrity, be described as follows:

Tab = Ea(Kab, B, A-stamp, KET, other attributes),

where A-stamp is a unique key stamp proving to A that the key is fresh and valid, KET is the key expiration time, and the "other attributes" might include such things as message type, encryption type, key type, key usage flags, etc. This indeed resembles very much the syntax of tickets used in Kerberos. If party identifiers (addresses or names) such as B, A-stamp, and other attributes are long, expressions such as this one result in long cryptographic messages, which lead to the packet size and export problems explained earlier. This ticket syntax assumes that the function E provides also integrity—there are encryption functions for which this does not hold, e.g., stream ciphers.

However basic security in primitive networking environments requires hardly any key attribute beyond the identifier of the party with which the key is associated and the key stamp and expiration time. Furthermore, as observed by the X9.17 designers, the only thing that really needs to be kept secret in a ticket is the key itself. The party identifier and key stamp may but need not be kept secret in principle. Thus a ticket syntax similar to the one used in X9.17 is sufficient:

(a) Tab = {Ea(Kab), B, A-stamp, KET}, MACa(previous 4 fields),

where the party identifier and key stamp are transmitted in cleartext but are tied to the key by the appended MAC field. This format helps export considerations because MAC implementations are exportable, and the usage of reversible encryption is restricted to the key itself.

However, for certain tickets, X9.17 also uses an even better (more compact) form of ticket, which it calls a notarized key. The notarization consists of folding the key attributes—in the X9.17 case, the party names and a unique key stamp derived from a counter—into the KEK that is used to encipher Kab. The resulting ticket format becomes simply

Tab = Ea*(Kab), B, A-stamp, KET

where the notarization key Ka* is a function of Ka, the party identifiers A and B, and the unique key stamp. X9.17 happens to use a relatively complicated function for combining the

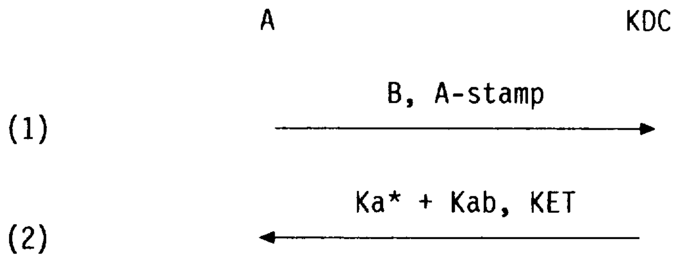


Fig. 2. Basic key distribution protocol.

party identifiers and key stamp into Ka*. However, looking at expression (a) above, one can observe that a simple yet secure way to hash these variables into Ka* is to define

(b) Ka* = MACa(B, A-stamp, KET),

where MACa is assumed to be pseudo-random, i.e., not to reveal any information about Ka* given all its arguments except Ka [25], [3]. The A-stamp proving the freshness of the key may be a unique time-stamp, key version number, or one-time random number (nonce) issued by the party A who requested the key. The resulting key Ka* is thus computable only by the KDC and by A, who are the only parties knowing Ka, and can thus safely be used to encipher Kab. Thus, the party identifier, key stamp, and KET can be transmitted fully in the clear, because their MAC integrity vector under key Ka is tied to Kab as part of the encrypted ticket used to transmit Kab.

This is one of the ideas behind the ticket syntax used in the family of server-based protocols to be described in the following section. Another idea is that encryption of Kab under Ka* may be done using any algorithm. DES would be fine if it is the local standard, but anything else will do as long as it is secure. One simple choice, available on any microprocessor, is to use Ka* simply as a one-time pad since it is pseudo-random, i.e. to compute its exclusive-or with Kab. This is less complex than DES, not to mention the fact that the exclusive-or operation is not subject to export controls. Under this assumption, tickets are simply of the form

Tab = (Ka* + Kab), B, A-stamp, KET,

where "+" indicates exclusive-or and Ka* is computed according to expression (b) above, using any suitable MAC algorithm, such as MD5 with Ka as a secret prefix for example [23], [25]. (Note that using a MAC as suggested above protects the secrecy of Kab but does not protect its integrity, in the sense that while an intruder cannot discover the key, it could change that key to an unknown value. This could be detected by an additional cryptographic integrity check but is not discussed here.) A corresponding ticket for the distribution of the same key to B would be

Tba = (Kb* + Kab), A, B-stamp, KET,

with Kb* = MACb(A, B-stamp, KET). When exclusive-or-based notarization is used, it is essential that each key stamp be unique, otherwise an intruder, having managed to obtain one key Kab, could derive Ka* and Kb*, and thus obtain subsequent keys shared by A and B, using the fact that Ka* and Kb* do not change for every Kab issued.

The resulting ticket syntax is thus exportable and leads to minimal-size messages. In practice, the key stamp may be just a nonce issued by the party that requested the ticket. Then, this nonce and the necessary party identifier, must already have been transmitted in cleartext as part of the original request, as illustrated in Fig. 2. Thus these parameters need not be unnecessarily repeated as part of the issued ticket; the ticket then boils down to just the KET and the notarized key itself, nothing else, thereby allowing minimal redundancy in message parameters, which in fact provides additional protection against potential known-plaintext attacks. Such economy of mechanism is not present in any of the existing, application-layer designs for key distribution (Kerberos, X9.17, X.509) for several reasons:

- the keys require many more attributes in such high-level environments;
- the designs make unnecessary use of encryption, enciphering many parameters that have nothing secret about them;
- key attributes are enciphered with the key rather than folded into it through notarization (except in X9.17);
- economy of mechanism was clearly not an objective in these designs.

With the foregoing discussion on the content and syntax of tickets behind us, we are in a position to describe the proposed family of flexible KDC-based protocols that avoid as much as possible all the limitations of existing application-layer designs mentioned earlier.

B. Scenarios and Protocols

A guiding concept behind our family of KDC protocols is its flexibility and adaptability to many different network configurations and connectivity patterns. In all coming scenarios, two entities A and B want to authenticate one another but have no key Kab initially. At the end of the scenario, the two entities have obtained a copy of the same shared secret key Kab and authenticated one another. (Any message they exchange subsequently can also be authenticated in the same way using the same key Kab.)

A-B-K Pull Scenario: Of all the scenarios and corresponding protocols to be described, the most easily understood one is called A-B-K (pull model); it resembles the X9.17 scenario in that the KDC is contacted by B. It is represented in Fig. 3.

In the A-B-K scenario, it is assumed that A either is unable, unauthorized, or unwilling to contact the KDC or it is willing to let B choose whether or not to contact the KDC.[2] Thus, according to the two-way authentication protocol suggested earlier, A contacts B in flow (1), challenging it to encipher a nonce Na. B then contacts the KDC in flow (1'), passing along A's identifier, A's nonce Na and its own identifier B and nonce Nb. The KDC replies in flow (2') by sending two tickets including the same fresh key Kab. The tickets follow the basic syntax suggested in the previous section, i.e., for Tab,

Tab = (Ka* + Kab), B, A-stamp, KET,

[2]For example, A and B may already share one key, but A defers to B for deciding whether to get another key or not.

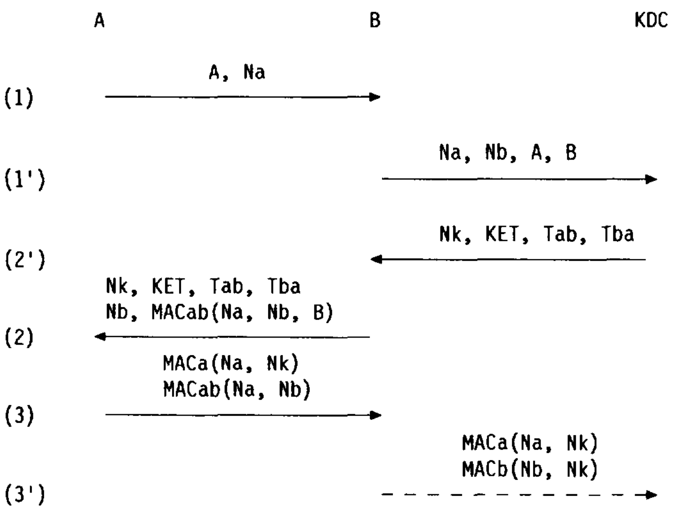


Fig. 3. A-B-K ticket distribution protocol.

where the A-stamp proving the freshness and validity of the key include one of three things:

- a simple but unique time-stamp, which would require clock synchronization as in Kerberos,
- a counter value, which would require counter synchronization as in X9.17, or
- a nonce associated with the requested key (i.e., Na in the present case).

As suggested in the previous section, the advantage of using counters or nonces is that these do not need to be transmitted explicitly as part of the ticket because they are already known from the context (in the case of counters) or were already transmitted as part of the request (in the case of nonces). One additional advantage of nonces is that they do not require the maintenance of any synchronized clock or counter, which is an essential consideration in situations where A and B may be simple link adapters without their own power supply or permanent storage. Another advantage is in scenarios involving multiple parallel connections between A and B, where race conditions between protocol initiations by A and B would make counter synchronization a major challenge. All protocols proposed here thus use nonces wherever possible. In addition to the two tickets, the KDC issues a nonce of its own, Nk, whose role will soon become clear.

Upon reception of the tickets, B may or may not cache them for reuse in later communication with A. The issue of caching tickets will be discussed soon. In any case B computes

Kb* = MACb(A, B-stamp, KET),

which it uses immediately to decipher Tba and retrieve Kab. In flow (2), B proceeds to compute and send to A its own nonce Nb and the expected authenticating expression

MACab(Na, Nb, B).

B piggybacks on the same message the values of Nk, Tab, and Tba that it received from the KDC. (Note that Tba is meaningless to A, since it is enciphered using B's KEK Kb. However A may want to cache it for later use.) In any case, upon reception of Tab, A computes

Ka* = MACa(B, A-stamp, KET)

and uses it to decipher and extract Kab from Tab. A then proceeds to reply to B with the expected authentication expression MACab(Na, Nb) in flow (3). For reasons explained below, A may also piggyback on that flow a similar expression, but computed using Nk instead of Nb and enciphered under Ka instead of Kab.

Upon reception of these messages, B has the option of computing a similar MAC expression but with Nb instead of Na and enciphered under Kb, and forwarding both MAC expressions to the KDC in an optional flow (3'). The result of that optional flow would be to acknowledge to the KDC that A and B received the intended key Kab and successfully authenticated one another. Such information may be useful to log for later auditing purposes. It is one of the reasons (besides security) for introducing Nk in flow (2'). Of course if such auditing is desired then the KDC needs to maintain state information associating Nk to A, B, and the newly distributed Kab.

This completes the description of the basic protocol. The issue of whether A and/or B should cache tickets depends on operational assumptions of the scenario at hand. If the scenario assumes that keys such as Kab are valid for one single interaction (message exchange or connection duration) between A and B, then tickets need not be cached, and keys should be stored and later erased as part of the interaction context information, thus forcing A and B to get new keys for each interaction. (In this case, the use of a KET parameter is also redundant.) On the other hand, if the KET allows spanning more than one interaction, A and/or B may want to cache tickets for the span of their lifetime: each can cache its own tickets knowing the other does the same, or one of them can cache both tickets knowing the other does not cache any. In either case, at least one copy of each ticket must be cached between the two partners, so that both can retrieve the key Kab if and when they resume interaction after a period of inactivity and within the lifetime of the key Kab.

Notice that, at the end of the whole exchange, A and B have acquired the same secret key Kab and authenticated one another. Yet the protocol can achieve more at very low cost: it can allow A and B to authenticate the KDC and vice-versa. Indeed, assuming that key stamps A-stamp and B-stamp are just nonces (Na and Nb), the original expressions for the notarization keys

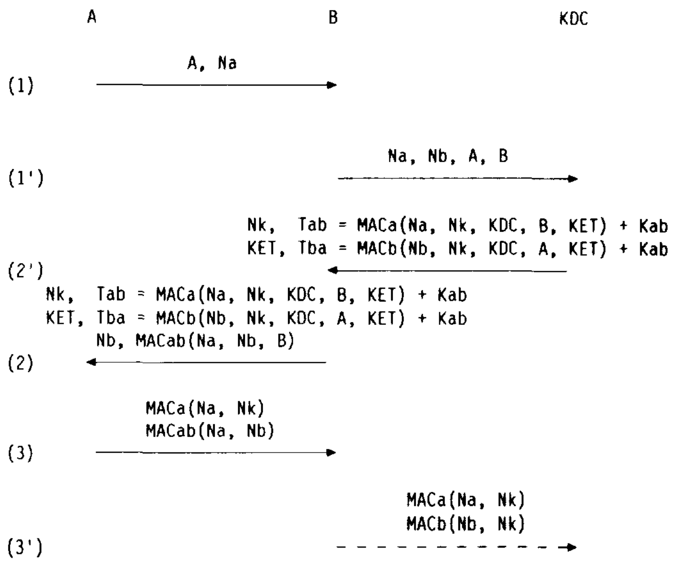
Ka* = MACa(B, Na, KET), and Kb* = MACb(A, Nb, KET)

simply tie together in one tamper-resistant MAC expression the KEK of one party to its nonce, the name of the other party, and the key expiration time. By replacing these two original expressions by

(c1) Ka* = MACa(Na, Nk, KDC, B, KET) and
(c2) Kb* = MACb(Nb, Nk, KDC, A, KET),

which include Nk, we achieve the same result but, in addition, the revised expressions match the canonical format and fulfill all necessary conditions to qualify as secure authentication tokens for the second flow of the basic authentication protocol, in the sense of [5] and [6].

Fig. 4 depicts the A-B-K scenario using the expanded syntax for both tickets. The use of this syntax is the main reason for introducing the nonce Nk. The revised expressions not only qualify as valid notarization keys to secure the necessary key distribution parameters but by doubling up as legitimate authentication tokens in the sense of the basic protocol, they allow A and B to authenticate the KDC. Thus, while the key distribution protocol achieves its main purpose and allows A and B to authenticate one another, it also achieves two-way authentication between the KDC and its clients because, in effect, the flows involving the KDC include, by superposition to the key distribution process, two separate runs of the basic authentication protocol, each run involving the KDC and one of its clients.


A
B
KDC
(1)
A, Na
(1')
Na, Nb, A, B
Nk, Tab = MACa(Na, Nk, KDC, B, KET) + Kab
KET, Tba = MACb(Nb, Nk, KDC, A, KET) + Kab
(2')
Nk, Tab = MACa(Na, Nk, KDC, B, KET) + Kab
KET, Tba = MACb(Nb, Nk, KDC, A, KET) + Kab
Nb, MACab(Na, Nb, B)
(2)
MACa(Na, Nk)
MACab(Na, Nb)
(3)
MACa(Na, Nk)
MACb(Nb, Nk)
(3')
Fig. 4. A-B-K ticket distribution protocol (expanded version).

In summary, the A-B-K scenario seen above amounts to nothing more than carrying out three instances of the basic two-way authentication protocol discussed earlier, one between A and B, one between B and the KDC, and one between A and the KDC (via B), where the latter two are slightly enhanced versions of the basic protocol to cater for key distribution. The resulting three-way protocol exhibits security, exportability, and minimal message overhead. The other scenarios in the remainder of this section are essentially only variations of this first one. They merely show how the KDC operation can dynamically adapt itself to various network configurations or connectivity patterns.

The foregoing discussion is of course intuitive, and a formal proof that the use of authentication tokens as notarization keys destroys neither the security of the authentication, nor that of the key distribution is essential. Some more arguments appear in Section IV-C while a formal proof is not the subject of the present paper. Similarly, with the above description of the protocol, it appears to be possible for an intruder or for B to change the value of Kab without A being able to detect it. In reality a judicious choice of algorithm for generating pseudo-random values for Nk can be used to bar that threat and guarantee the integrity of the key. This is however also beyond the scope of the present paper, and is addressed in a separate publication [13].

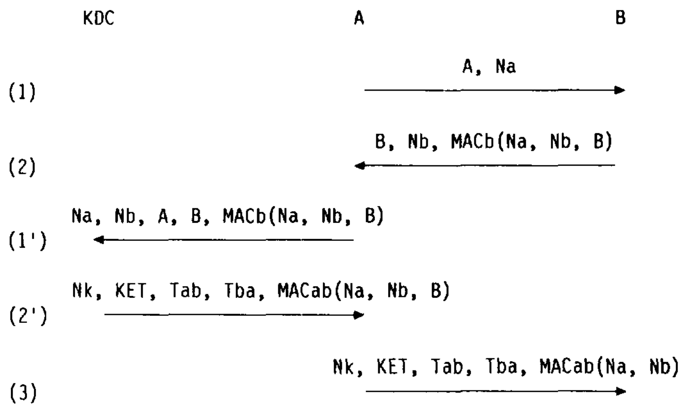

KDC
A
B
(1) A, Na
(2) B, Nb, MACb(Na, Nb, B)
(1') Na, Nb, A, B, MACb(Na, Nb, B)
(2') Nk, KET, Tab, Tba, MACab(Na, Nb, B)
(3) Nk, KET, Tab, Tba, MACab(Na, Nb)

Fig. 5. K-A-B ticket distribution protocol

K-A-B Push Scenario: In this second scenario, called K-A-B (push model), the KDC is contacted by A instead of B, as shown in Fig. 5. A contacts the KDC because B is either unable, unwilling or not allowed to do so.

The scenario starts with the same flow (1) in which A challenges B to encipher some nonce Na. Since B, for whatever reason, does not contact the KDC as in the previous scenario, it does not have a key Kab to authenticate itself to A. Thus it uses its main KEK Kb instead, and omits in flow (2) to return any ticket to A as it would have done in the A-B-K scenario.

Not seeing any tickets in (2), A deduces that B did not contact the KDC and thus (unless it has cached tickets from a previous interaction with B) proceeds to contact the KDC in step (1'). There, it passes to the KDC its own nonce as well as B's identifier and B's nonce Nb. It also appends B's authentication token

MACb(Na, Nb, B)

since it does not know Kb and thus cannot check that token.

In flow (2'), the KDC recognizes a K-A-B scenario and thus checks B's token for A, replies with Nk, KET, Tab and Tba as usual, and appends B's token translated from key Kb to the new key Kab to inform A that B's original token did "check" all right (under the assumption that Na is fresh, which A knows is true). The tickets have the same format as in the A-B-K scenario.

At that point, A may or may not cache the tickets for later use, and computes Ka* to decipher and extract Kab from Tab. It then passes Nk, KET, Tab and Tba and the usual token MACab(Na, Nb) to B in flow (3) to complete the two-way authentication. B may or may not cache the tickets, computes Kb* to decipher and extract Kab from Tba, and verifies A's token.

Notice that this scenario as it stands does not allow A and B to complete a two-way authentication protocol with the KDC. They implicitly authenticated the KDC but the KDC could not authenticate them without a couple of extra flows, which would be substantially more costly and harder to map on typical connection set-up protocols. To that extent the use of Nk in the tickets does not serve any particular purpose. Nk is used here simply to preserve some homogeneity between ticket formats in all scenarios, but for no other significant purpose.

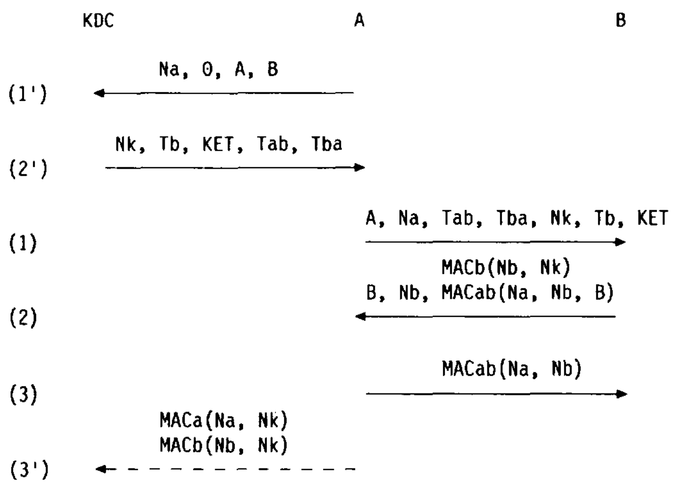

KDC
A
B
(1') Na, 0, A, B
(2') Nk, Tb, KET, Tab, Tba
(1) A, Na, Tab, Tba, Nk, Tb, KET
MACb(Nb, Nk)
(2) B, Nb, MACab(Na, Nb, B)
(3) MACab(Na, Nb)
MACa(Na, Nk)
MACb(Nb, Nk)
(3')

Fig. 6. K-A-B(t) ticket distribution protocol.

K-A-B(t) Push Scenario

In certain network scenarios, it may be desirable to use another push protocol, referred to as K-A-B(t) and represented in Fig. 6, to force A to get the tickets before communicating with B, or to prevent B from contacting the KDC. This protocol exhibits a Kerberos-like flow.

In this scenario, A contacts the KDC directly with its nonce Na and B's identifier in flow (1'). Notice that A cannot provide the KDC with a nonce for B since it has not yet communicated with B. Seeing that Nb is absent (e.g., null), the KDC infers a K-A-B(t) scenario. In this case, it replies in flow (2') with the usual Nk, KET, Tab and Tba. However, since Nb was absent, it cannot be used as the freshness stamp in Tba. Instead a time-stamp Tb **must** be used in lieu of Nb in this case. Notice that the use of such a time-stamp does not require that all parties keep clocks synchronized as tightly as in Kerberos if that time-stamp is used only to prove the freshness of the key and not to authenticate the KDC to B. (This latter task would have required a unique time stamp.) A time-stamp that is loosely synchronized (e.g., to the nearest 5 min tick) would do in the present case.

Upon receiving the tickets, A may cache them, derives Kab as usual, and proceeds to send its nonce Na, together with Nk, Tb, KET, Tab and Tba to B in flow (1). Seeing two tickets coming in on the first message, B deduces a K-A-B(t) scenario. It may cache the tickets, derives Kab from Tba, after checking that Tb is within the allowed time window (or any agreed upon validity check required for Tb), and replies in flow (2) with the usual MACab token for A. It may optionally add a MACb(Nb, Nk) token for the KDC.

A receives and checks the MACab token from B, replies with its own token on flow(3), and then may forward the optional MACb token to the KDC, together with its own MACa to complete the scenario with the optional flow (3') and allow the KDC to authenticate A and B and log that they successfully authenticated to one another.

C. Arguments for the Security of the Protocols

We now give intuitive arguments for the security of the proposed key distribution protocols. As mentioned earlier, a formal definition and analysis of security is beyond the scope of this paper.

The security of the protocols assumes the security of the underling cryptographic primitives. That is, any attack which breaks the protocol implies an attack which breaks the cryptographic primitives. Thus, there is no weakness attributable to the protocol itself.

In the proposed protocols the relevant cryptographic primitive is the MAC function (which uses random private keys shared between the involved parties to achieve secrecy). We adopt a common cryptographic assumption, which is that the MAC function is pseudo-random. This assumption implies the following property: even having seen pairs of the form (arguments, MAC of the arguments) an attacker with no knowledge of the shared secret key still has no knowledge of the results of the MAC when applied to new arguments. (This property is required for instance to protect the secrecy of a key through exclusive-OR with a MAC.)

Somewhat more formally: assume that every attacker can only perform polynomial time computations. The MAC function has the property that such an attacker will fail in the following attack. The attacker cannot distinguish between two black boxes when one box contains a truly random function (that gives random values given arguments) while the other box contains the MAC function with an unknown random secret key.

We note that our protocols do not use the standard CBC DES MAC function because it is not pseudo-random when applied to variable-length strings. The MD5 function, applied to a string concatenated with a secret key, may be viewed as suitably pseudo-random. (See the arguments proposed in our own work [5] and developed subsequently in [3]).

A key distribution protocol is secure if the distributed key remains secret. Namely, whenever a party A accepts key Kab as a session key, then this key is unknown to any adversary. A stronger condition is to require the key Kab to be pseudo-random, namely any polynomial time adversary cannot distinguish between the key Kab and a random string. We argue that the keys accepted by parties from the proposed protocols appear pseudo-random to any polynomial time adversary.

The security follows from considering the entire set of expressions that are available to an adversary from runs of the protocol, and the keys that a party may accept.

Consider a run where party A, having selected a random nonce Na, accepts key Kab for a session with party B. In all the proposed protocols this implies that party A received a string

MACa(Na, Nk, KDC, B, exp) + Kab

where exp is any expression. We first claim that the string

MACa(Na, Nk, KDC, B, exp)

is pseudo-random. This follows immediately since A selects Na randomly at the beginning of the session, and therefore it (with overwhelming probability) never calls the pseudo-random function MACa with the same set of inputs. Similarly, KDC selects Nk randomly, and therefore never calls MACa with this set of inputs.

It follows that an adversary which is neither A nor B does not gain any useful information from having both

MACa(Na, Nk, KDC, B, exp) + Kab and
MACb(Nb, Nk, KDC, A, exp) + Kab.

One may think that the adversary may learn something about Kab from the other flows which involve that key. Note that Kab is either selected randomly by the KDC, or appears pseudo-random to the adversary. The only other expression where Kab may appear is when a party sends

MACab(Na, Nb, B) or MACab(Nb, Na, A).

But since we assumed the MAC function is pseudo-random, it follows that this does not give any information about Kab either.

The same holds even if on past executions for which the adversary may have somehow learned the actual key (thus learning the value of the MAC on the corresponding nonce). The learning of past keys (MAC values) does not help in learning a current one as a party (say the KDC) always chooses a new value for it.

So far we have envisioned an adversary who observes the flows. Now consider the case of an adversary actively trying to send messages (an impersonator). Even in this case the impersonator does not control the nonce values that the MAC function is applied to. Thus the results still appear pseudo-random, and further they are hard to produce without the secret keys Ka, Kb, and Kab. Thus this impersonator cannot produce the required answers (with significant probability). Thus, trying to break the protocol and even running many instances of it cannot help an active intruder. (The arguments to such claims are derived from the ones in [5] where the notion of interleaving attacks and its more formal notion of matching histories and successful attack were put forth.)

In general the combination and formalization of the above arguments constitute the foundation for the security of the protocol and the fact that active and passive attacks combined cannot make a party impersonate an entity or a security server (unless the underlying cryptographic function can be compromised). The formal arguments do not simply use the security of the basic authentication protocol to argue that the distribution is secure as well; the key distribution and subsequent authentications are proven secure together against any combination of passive and active attacks.

V. Discussion

The three KDC protocols presented above form a consistent family that can all coexist and be supported simultaneously by the KDC. The KDC and each party can easily discover dynamically, at run-time what scenario to play based on the other parties' behavior. The existence of so many different scenarios makes it conceivable that two parties A and B could start behaving in incompatible ways, e.g., both insisting that they should contact the KDC or both declaring that they cannot do so. The resolution of such deadlock situations calls for architectural rules specifying either that one of the scenarios is a default that all implementations must agree to support as a minimum option, or alternatively that some implementations may not be able to inter-operate because of incompatible assumptions or operational conditions. The KDC of course supports all options.

All protocols together avoid the limitations of existing designs in that they allow any scenario, and the ticket format they use avoids reliance on tightly synchronized clocks or counters, minimizes message sizes, minimizes cryptographic operations, and stays clear of export restrictions through use of plain MAC operations, except for the key encryption operation itself, whose exclusive-or-based concept is not directly suitable for efficient bulk encryption, is therefore not subject to export controls.

A protocol variant in which A and B use key distribution to refresh a key Kab they share can be easily derived from the protocols above. In fact A and B can use Kab to exchange another key Kab', and then use Kab' to perform two-way authentication or other functions.

One may legitimately ask why the proposed protocols do not minimize the number of messages. Other designs have been proposed that require only four messages, e.g., [24], [21], [14], [20] instead of the five or six shown in our protocols. It is indeed possible to marry the light-weight mechanisms and one-way functions used in our designs with the flow diagrams used in these other designs to define an enhanced family of protocols, some of which would have only four flows. However, doing so would require assuming either tight clock synchronization as in Kerberos [24], and/or full connectivity between all involved parties as in [21], [14], and [20]. Both of these assumptions can of course be made in theory. They were not made in our case because they were not felt justified in environments of the sort we envisioned, namely low-function link or network protocols. Indeed, in these cases not all participants have the means and the connectivity required for supporting the reduced flows. In fact, even if they had, the protocol flows assumed in [21], [14], and 20] may be tricky to implement in practice as they require sharing parameters (e.g. nonces) between different connection contexts since some of the request-response message pairs they include are sent and received on different links or connections. This can certainly be implemented but is at the very least unusual for communication protocols, and may be hard to map on some existing implementations. Finally it should be pointed out that the 6th flow in some of our protocols is always optional, but—if present—adds a function not present in many of the four-flow designs, namely authentication of the clients to the KDC.

VI. Conclusion

In this paper, we have described a family of KDC-based key distribution protocols catering to different network connectivity patterns. All the proposed mechanisms use concepts from the same basic two-way authentication protocol and extend them to the distribution of tickets.

The resulting family of protocols differs from previous designs in that it is specifically tailored towards low-function networking environments, where good security is required but must be provided with minimal overhead, must cater to various restrictive network configurations and connectivity patterns, and must be exportable, which is achieved by relying solely on the use of one-way hash functions for all authentication and key distribution protocols, as observed independently in [10]). The simplicity, and economy of the proposed protocols means that they can be used for building security into even the most primitive devices and networking components, thus allowing these to participate fully and securely in future networks.

All the protocols in this paper have now been implemented as part of a comprehensive security subsystem prototype called KryptoKnight, whose software and implementation aspects are discussed in [16], and which formed the basis for the IBM network security program product.

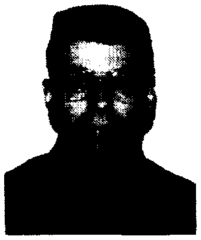
Acknowledgment

The authors wish to thank L. Svobodova and G. Tsudik for their comments on drafts of this paper, which helped focus the discussion, clarify explanations and improve the style. They also thank the reviewers of the IEEE/ACM Transactions on Networking for their comments which greatly helped in shaping the paper to its present form.

References

[1] R. K. Bauer, T. A. Berson, and R. J. Feiertag, "A key distribution protocol using event markers," *ACM TOCS*, vol. 1, no. 3, pp. 249-255, 1983.
[2] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Proc. Crypto'93*; also to appear in *Advances in Cryptology*, Springer-Verlag.
[3] ——, "Random oracles are practical: A paradigm for designing efficient protocols," *Proc. 1st ACM Conf. Comput. Commun. Security*, Nov. 1993.
[4] S. M. Bellovin and M. Merritt, "Limitations of the Kerberos authentication system," *ACM CCR* vol. 20, no. 5, pp. 119-132, Oct. 1990.
[5] R. Bird *et al.*,"Systematic design of two-party authentication protocols," in *Proc. Crypto 91*, Santa Barbara, CA, Aug. 1991, pp. 44-61; also available as "Advances in cryptology," in *Lecture Notes in Computer Science*, J. Feigenbaum, Ed. New York: Springer-Verlag, 1991, vol. 576.
[6] ——, "Systematic design of a family of attack-resistant authentication protocols," *IEEE J. Select. Areas Commun.*, vol. 11, pp. 679-693, June 1993.
[7] D. E. Denning and G. M. Sacco, "Timestamps in key distribution systems," *Commun. ACM*, vol. 24 no. 8, pp. 533-536, Aug. 1981.
[8] Data Encryption Standard, FIPS 46, NBS, Jan. 77.
[9] A. Herzberg, S. Kutten, G. Tsudik, and M. Yung, "Designing efficient key exchange protocols," unpublished.
[10] L. Gong, "Using one-way functions for authentication," *ACM CCR* vol. no. 19, no. 5, pp. 8-11, Oct. 1989.
[11] Banking—Key management (wholesale), ISO 8732, Geneva, Switzerland, 1988.
[12] OSI Directory—Part 8: Authentication framework, ISO 9594-8, Geneva, Switzerland, 1988.
[13] P. Janson and G. Tsudik, "Secure and minimal protocols for authenticated key distribution," IBM Zurich Res. Lab., Res. Rep. RZ 2538, Dec. 1993; also to appear in *Comput. Commun. J.*
[14] A. Kehne, J. Schoenwaelder, and H. Langendoerfer, "A nonce-based protocol for multiple authentications," *ACM OSR*, vol. 26, no. 4, pp. 84-89, Oct. 1992.
[15] T. Lomas, L. Gong, J. Saltzer, and R. Needham, "Reducing risks from poorly chosen keys," in *Proc. 12th ACM SOSP, ACM OSR*, vol. 23, no. 5, Dec. 1989, pp. 14-18.
[16] R. Molva, G. Tsudik, E. Van Herreweghen, and S. Zatti, "KryptoKnight authentication and key distribution system," in *Proc. ESORICS 92*, Toulouse, France, Nov. 23-25, 1992).
[17] R. Morris and K. Thompson, "Password security: a case history," *Commun. ACM*, vol. 22, no. 11, pp. 594-597, 1979.

[18] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, no. 12, pp. 993-998, 1978.
[19] ——, "Authentication revisited," *ACM OSR*, vol. 21, no. 1, p. 7, Jan. 1987.
[20] B. C. Neuman and S. G. Stubblebine, "A note on the use of timestamps as nonces," *ACM OSR*, vol. 27, no. 2, pp. 10-14, Apr. 1993.
[21] D. Otway and O. Rees, "Efficient and timely mutual authentication," *ACM OSR* , vol. 21, no. 1, pp. 8-10, Jan. 1987.
[22] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key crypto-systems," *Commun. ACM*, vol. 21, no. 2, pp. 120-126, 1978; also in *Commun. ACM*, vol. 26, no. 1, pp. 96-99, 1983.
[23] R. L. Rivest, "The MD5 message digest algorithm," Internet draft, RSA Co., July 91.
[24] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication server for open network systems," *in Proc. Usenix Conf.*, Winter, 1988.
[25] G. Tsudik, "Message authentication with one-way hash functions," *Proc. IEEE INFOCOM 92*, Florence, Italy, May 1992.



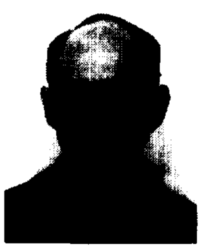
Ray Bird received the B.S. degree in mathematics from Rutgers Unversity, NJ.

He is a Senior Engineer in the Networking Systems Architecture group at IBM, Research Triangle Park, NC. He joined IBM in 1967 and until 1977 he worked on the OS/360 operating system and the Telecommunications Access Method (TCAM). From 1977 to the present he has worked on Systems Network Architecture protocols including subarea, LU 6.2, and Advanced Peer-to-Peer Networking (APPN). He is currently working on future enhancements to APPN.

Inder Gopal (F'91) received the B.A. degree in engineering science from Oxford University, Oxford, England, in 1977, and the M.S. and Ph.D. degrees in electrical engineering from Columbia University, NY, in 1978 and 1982, respectively.

Since 1982, he has been with the IBM Corporation serving in various technical and management positions. Currently, he is Division Director of Architecture in the Networked Applications Solutions Division. Previously, he was Manager of the Advanced Networking Lab at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. In that capacity he was involved in the NSF/DARPA sponsored AURORA Gigabit testbed and in several other research projects in the area of high-speed networking. His other research interests are in distributed algorithms, communication protocols, network security, high-speed packet switches, and multimedia communications. He has published extensively in these areas. He has received an Outstanding Innovation Award from IBM for his work on the PARIS high speed network. He is an Editor for the *Journal on High Speed Networks* and has previously served as an Area Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS, Guest Editor for *Algorithmica*, Guest Editor for IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, Editor for Network Protocols for the IEEE TRANSACTIONS ON COMMUNICATIONS, and Technical Editor for IEEE COMMUNICATIONS MAGAZINE. He has served on several program committees for conferences and workshops.

Amir Herzberg photograph and biography not available at the time of publication.



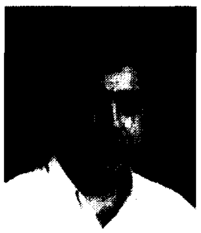
Phil Janson received the B.S. degree in electrical engineering from the University of Brussels, Belgium, in 1972, and the M.S., E.E., and Ph.D. degrees in computer science from the Massachusetts Institute of Technology, Cambridge, in 1974, 1975, and 1976, respectively.

Since 1977, he is with the IBM Zurich Research Laboratory, Switzerland, where he has been working on high-speed packet switches, the IBM Token Ring, and LAN servers and gateways. During 1986-1987, he was on assignment at the IBM Development Laboratory in Austin, TX, working on LAN gateways for OS/2. Since 1987 he has been managing various research projects dealing with the integration of heterogeneous networks, and more recently network security. He is a Visiting Professor at the Department of Engineering of the Free University of Brussels where he has been teaching an Operating Systems course since 1976. He has also taught various courses on operating systems, LAN's, OSI, and network security at IBM, at various international conferences, and for different professional organizations. His research interests include operating systems, distributed systems, computer communications, and their security. He is the author of many papers and patents in these fields, as well as a textbook on operating systems.

Dr. Janson received a Harkness Fellowship in 1972 and several IBM invention and technical awards since then. received a Harkness Fellowship in 1972 and several IBM invention and technical awards since then. He has been a member of the ACM since 1974, and an affiliate member of the IEEE Computer Society.

Shay Kutten (M'90) received the Master's degree (on scheduling of radio broadcasts) and the Ph.D. degree (on distributed network algorithms) in computer science from the Technion—Israel Institute of Technology, in 1984 and 1987, respectively.

Since 1987 he has been with IBM Thomas J. Watson Research Center, Yorktown Heights, NY, as a Post-Doctoral Fellow, as the Manager of the Network Architecture and Algorithms group, and as a Research Staff Member. He has developed algorithms for network control, security, and distributed processing control, that were incorporated in IBM's products.



Refik Molva (M'87) received the Ph.D. degree in computer science from the Paul Sabatier University, Toulouse, France, in 1986

After five years as a Research Staff Member at the IBM Zurich Research Laboratory, he has been an Associate Professor at the Eurecom Institute, Sophia Antipolis, France, since 1992. His research interests include network security and communications for multimedia applications.

Moti Yung photograph and biography not available at the time of publication.