Scalability and Flexibility in Authentication Services: The *KryptoKnight* Approach

Philippe Janson* Gene Tsudik† Moti Yung‡

Abstract

This paper studies the issues of *flexibility* and *scalability* in the context of network security. In particular, it concentrates on authentication and key distribution services suited for a variety of communication paradigms, network environments, and end-devices. We present the design criteria, specification, and step-by-step construction of authentication and key distribution services based on experience in the *KryptoKnight* project.

The central goal of the *KryptoKnight* project was the construction of basic network security functions in a *minimal*, *flexible* (thus, versatile) and *scalable* manner. Protocol minimality (in terms of resource usage) and flexibility are not merely theoretical goals; they have clear advantages in environments where computational resources are limited and connectivity is restricted. *KryptoKnight* was aimed at such environments: small and *anemic* wireless devices, simple network and data-link entities, embedded micro-devices and other special-purpose communication equipment and configurations. Furthermore, *scalability* of protocols makes their deployment possible in the presence of rapid network growth and inter-domain communication.

1 Introduction

The objective of the *KryptoKnight* project was the design of minimal, flexible and scalable authentication and key distribution protocols for various network settings such as low-level networking environments (e.g., link and network adapters) and small devices with limited computing power (e.g., PDAs and mobile telephones). Such components increasingly find their way into various areas, e,g., office, field, home, entertainment and education. Moreover, resarch on "ubiquitous computing" is advancing rapidly [10]. Unlike a workstation in a client-server setting, these limited-ability devices still require security features: device authentication to prevent wireless fraud, calling and called address identification to keep intruders out, low-level authentication to secure booting from a LAN adapter and so on. Security functions, in turn, typically require cryptographic support and, thus, key distribution protocols. However, integration of security features into primitive components must take into account restrictions such as: small packet sizes, low bandwidth, high transmission costs, limited processing and storage resources, real-time constraints and export regulations.

Under these conditions, designing simple, secure and versatile mechanisms is not obvious. Solutions must be flexible, i.e., suitable for various network configurations and connectivity patterns. Solutions must *fit* into devices with a wide range of computational capabilities, including: palm-top, lap-top, desk-top, set-top, or floor-top devices. Moreover, the most primitive devices must be able to inter-operate securely with the most powerful ones, which implies the need to provide common security functions to meet the minimal needs. Furthermore, solutions must be scalable so as to accommodate growth both within and across interconnected administrative domains.

In general, achieving minimal and flexible design is desirable because of the prevailing view that security mechanisms are expensive, performance-impairing, and cumbersome. (Since security mechanisms typically require complex administration and their deployment is subject to limitations such as export control or licensing.) This often results in users and organizations avoiding, disabling, or downgrading security mechanisms.

Finally, minimality of mechanism really pays off whenever security features must be *retrofitted* into existing, established protocols or applications. If, as is often the case, security features are introduced as an afterthought, security parameters have to be squeezed into the meager space available in memory or packet headers.[1]

In this paper we review the *KryptoKnight* protocol family which was designed with the explicit goals of flexibility, minimality and scalability. Early aspects of this work have been already reported on and have had their impact. The basic 2-party protocol [5] influenced the ISO basic authentication standard [17], and [7]–the ISO key management standard. The early work also initiated a research topic of proof correctness in cryptographic protocols based on matching of protocol histories and defending against replay and interleaving attacks [5, 11, 3, 4].

This paper, on the other hand, discusses other issues experienced and learned in the course of the *KryptoKnight* project. In particular, we consider the issues of flexibility and scalability in both intra- and inter-domain protocol design. More specifically, we present the rationale for the design concepts and develop the initial design of the protocol family geared towards various communication scenarios. Then, we concentrate on the inter-domain protocols that provide scalability of the overall architecture and show how the basic protocols' properties enable modular design of their inter-domain counterparts.

Organization: Section 2 addresses some background issues and the basic protocol building blocks. In Section 3, the optimized key distribution protocols are derived and a set of 3-party protocol scenarios and protocols is presented. Section 4 discusses the requirements for, and design of, the inter-domain protocols. Section 5 concludes with the summary of this work.

*IBM Research, Rüschlikon, Switzerland; *pj@zurich.ibm.com*.
†USC/ISI, Marina del Rey, CA; *gts@isi.edu*.
‡CertCo, New York, NY; *moti@certco.com*.

[1] This point is best illustrated by considering what it takes to "secure" IP, Telnet or ATM networks.

2 Basic Concepts and Preliminaries

A number of schemes have been designed for authentication and key distribution services in distributed systems and networks, e.g., [25, 8, 1, 26, 29, 21, 27]. Some of these have lead to implementations or standards, e.g. the Kerberos [32, 28] and ANSI X9.17 [18] schemes, which use DES technology, or the ISO-CCITT X.509 Directory Authentication scheme [19], which relies on public key technology. Kerberos is the basis for the security component provided by the Open Software Foundation (OSF) as part of its Distributed Computing Environment (DCE) offering. The *KryptoKnight* concept and resulting architecture differ from the above mainly owing to the specific computing restrictions and the general flexibility/scalability requirements we next discuss. As a private-key based system, a scalable design must be based on trusted servers [25], which assure that various parties are able to undergo mutual authentication without initially sharing cryptographic keys and based solely on the established trust with the server(s).

2.1 Basic Concepts and Design Issues

Before delving into the actual protocols, we begin (in the next subsection) with the review of the key design issues.

2.1.1 Minimal Resource Utilization

Keeping in mind resource-poor, restricted network environments, our main requirement is to minimize bandwidth usage and computational overhead. This translates into small messages, small cryptographic tokens, few cryptographic operations and minimal storage at the device. This economy of mechanism rules out using public key designs (ISO-CCITT X.509 Directory). Even Kerberos with its relatively verbose tokens and liberal use of encryption may be too costly. Of course, our approach implies a certain trade-off with the semantic richness of cryptographic tokens in designs like X.509 or Kerberos. However, such richness is typically unnecessary at the basic level of communication protocols and devices that we have in mind.

We note that constructing cryptographic primitives on top of basic functions is an on-going research effort in theoretical cryptography (e.g., digital signature schemes have been shown to require only one-way functions [24, 31]). We concentrated on building key distribution and authentication primitives in the trusted server model without the full capacity of encryption/decryption functions but, with only pseudo-random one-way functions.[2] In addition, we minimize the use of cryptographic operations and the number of message flows in order to minimize the software that the resulting protocols require. Related minimization ideas have been independently developed by Gong [14, 15].

2.1.2 Freedom from Export Restrictions

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 move from country to country, the situation can easily become intractable if authentication protocols were restricted. For example, how can one deal with a situation where a user from country A visits country B and communicates with a user in country C, using equipment produced in country D? Such problematic issues can only be resolved with the availability of a universally-accepted technology which is free of any governmental limitations. Lastly, experience with the design of cryptographic systems shows that maintaining different system versions – exportable and non-exportable – can be quite costly. There are a number of ways to assure exportability, by employing weakened cryptography (e.g., a 40-bit DES keys), or tight encapsulation of the crypto module. In our design we employ a well defined cryptographic module that employs only a one-way function rather than a block-cipher; this limits the applicability of our cryptography for bulk data encryption (which is the major concern for governments). In addition, modularity may help in careful design for exportability. For example, using different kinds of cryptographic tools at different levels.

[2] In practice, typically a keyed cryptographic hash suffices.

2.1.3 Flexible Operational Scenarios

Most existing key distribution methods have been tailored to the relatively narrow operational scenarios of specific distributed applications. 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 reliable clocks in a Wide Area Network (WAN) is conceivable, it is certainly harder.

More importantly, existing schemes make specific assumptions about network configuration and connectivity models. For instance they may dictate a specific communication paradigm for contacting a trusted server or key distribution center (KDC): 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 a LAN environment – for which Kerberos was initially designed – requiring clients (rather than servers) to obtain the keys makes sense because it distributes the burden over many clients, thus alleviating the task of the few shared servers. In a WAN environment – for which X9.17 was designed – the opposite approach is justified because there are typically many more clients than servers and KDCs are typically located closer to servers than to clients. Under such circumstances, the amount of system definition (configuration) and the cost of client-KDC connections required by the Kerberos approach could become prohibitively high in a WAN context. In conclusion, dictating a fixed connectivity pattern is possible in well-defined application environments with clear operating assumptions. However, it becomes unrealistic in general network environments. In the setting of a wireless-network, clients contact communication servers (base stations) at their entry point to the network, the pull model is mandatory in such a case.

2.1.4 Scalable Design

Closely related to flexible operation is the notion of scalable design. Networks are often interconnected and portions thereof are administrated by different authorities. A general authentication and key distribution solution must be extensible to large internetworks comprised of many interconnected domains. Flexibility in connectivity and in access pattern to the security servers eases and enhances scalability, since it allows different clients and servers (under various mobility scenarios) to establish authenticated sessions. For this reason, the *KryptoKnight* design attempts to cover a number of different inter-domain environments. This is in contrast with the Kerberos inter-domain model which puts the burden on the client by making it responsible for searching and accessing the appropriate security server outside its own domain.

2.2 Basic Building Block: 2-Party Authentication

A family of simple 2-party authentication protocols was presented in [5, 6]. The protocols are efficient in message size and computation overhead and minimal in use of cryptography (e.g., can use a block cipher or even a one-way hash function employing a common key as a private portion of the message). They were shown resistant to a wide set of attacks known as interleaving attacks. In addition, essential ideas for formal model and proofs of correctness for these and similar protocols were suggested initially in [5] and further developed in [11, 3]. Simple design eases these complexity-theoretic proofs. Conceptually, we attempted to have a uniform design basing our protocols on similar basic operation; this eases the design's conceptual complexity and its validation, while making it widely (and incrementally) applicable to numerous scenarios.

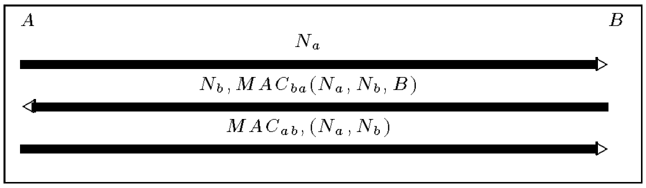


Figure 1: 2-Party Mutual Authentication Protocol (2PAP)

One of the protocols derived from the work of [5] is called 2PAP and is represented in figure 1. In this figure, the letters A and B stand for the identifiers of two network entities performing mutual authentication. The variables N_a and N_b are one-time random numbers (called nonces) used by each party to challenge the other to prove its identity. The notations MAC_{ba} and MAC_{ab} denote cryptographic one-way hash functions, called *Message Authentication Codes*, computed with keys K_{ba} and K_{ab} respectively, and used to guarantee the authenticity of their parameter string. (The commas between the MAC parameters indicate different blocks to which the MAC function is subsequently applied, e.g. as in CBC mode of block ciphers. Namely the key is applied to the first block the result is ex-ored with the next block and the key is applied, and so on..). Implementing MAC functions with a symmetric cryptographic system such as DES [9] (CBC mode), K_{ab} and K_{ba} are (typically equal) secret keys shared by A and B. Instead of using an encryption function to implement the MACs, one can also use a plain one-way hash function (e.g. the MD5 algorithm employed on fixed size blocks [30] is believed to be such a function, another function is SHA1) applied a number of times in some way to the authenticated string concatenated with the secret key (see, e.g., [33]). Using a plain one-way hash function (under the assumption that it is pseudo-random [13]) 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 (i.e., one can argue that using this function for *bulk data encryption* is either highly inefficient or requires additional cryptographic support). (As a matter of practicality, our designs, in fact, have gotten export license).

In addition, the protocol uses MAC operation twice (or three times if we count block encryptions in CBC mode.) The number of message flows (three) is *minimal* for a nonce-based protocol. Furthermore, the combined size of all cryptographic messages in terms of data blocks (four) is also *minimal*. The use of nonces is robust since nonces are typically generated with cryptographic means and require no *state* maintenance as opposed to alternative methods that rely on synchronized clocks or counters. This basic protocol serves as the foundation for the design of the entire *KryptoKnight* protocol family discussed in this paper.

3 Server Based Key Distribution

Key distribution by a server to users is central in our architecture. We now proceed to discuss key distribution issues and their various shapes, forms and flavors. After introducing the notation and discussing some preliminaries, we go into 2-party and (server-based) 3-party protocol scenarios.

3.1 Notation

A,B,C,X,Y	principals
K_x	KEK of X (known only to KDC and X)
K_{xy}	Secret key shared by X and Y
$E_x(M)$	Encryption of M under key K_x
$E_{xy}(M)$	Encryption of M under key K_{xy}
$MAC_x(M)$	MAC of message M under key K_x
$MAC_{xy}(M)$	MAC of message M under key K_{xy}

3.2 Tickets

Distribution of secret keys from the KDC to its constituent principals (clients and servers) requires the use of secret channels. In the realm of symmetric cryptography this requires that each principal in the network share at least one secret key with the KDC. This key is often referred to as the principal's *key-encryption key* (KEK).

The nature of the messages exchanged between a KDC and its clients is reminiscent of public key certificates: a key distribution expression issued by the KDC (also referred to as a *ticket*) must, by definition, include the new key. It must also include a *freshness indicator* (such as a unique key version number, time-stamp, or nonce) indicating to the recipient that the key is fresh and valid. In addition, a ticket must include the identifier(s) of the principal(s) for whom the key is intended. It may also include such information as key expiration time and other usage parameters.

3.3 Basic Key Distribution

Drawing on our discussion above, a simple and secure 2-party key distribution protocol can be easily constructed from 2PAP (as depicted in Figure 2.)

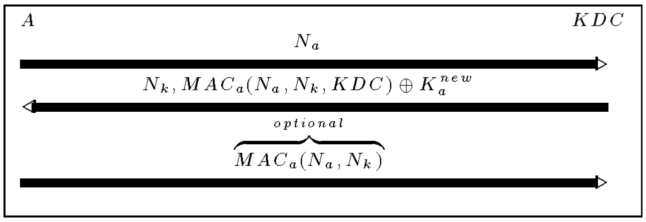


Figure 2: Basic 2-Party Key Distribution Protocol (2PKDP)

Unlike its pure authentication counterpart, 2PKDP consists of only two flows. The third flow is not usually needed since it would require the KDC to keep state between flows 2 and 3.

For simplicity's sake, the expiration time and other usage-related parameters are omitted since they are likely to represent constant or seldom-changing values. For example, a validity interval of 8 hours may be globally agreed on by default, or it may be included in ticket construction (i.e. its value may be added to the arguments being MAC-ed).

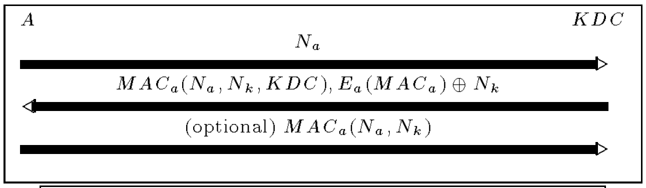
The resulting ticket syntax is minimal and exportable. It yields simple and short messages, and exhibits minimal redundancy in message parameters. Such economy of mechanism is certainly not present in any of the existing, application-layer designs for key distribution (Kerberos, X9.17, X.509). This may be due to several reasons:

- Keys requiring many more attributes (semantics) in such higher-layer environments,
- The designs make unnecessary use of encryption, encrypting many parameters that have nothing secret about them (in many cases, the notion of authentication is mistakenly associated with encryption),
- Key attributes are encrypted 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.

3.4 Authenticated Key Distribution

It can be easily seen that the 2PKDP provides key distribution and nothing else. The protocol is secure in the sense of key confidentiality, i.e., no one other than A and the KDC is able to obtain K_a^{new}. (The security of this protocol is discussion in detail in [34].) However, it is clearly not secure with respect to key integrity. This means that any intruder can modify the key distribution and cause A to extract a key *not issued* by the KDC. In fact, the KDC may be unavailable and the entire second flow can be concocted by the intruder. We note that the original design of the protocol family relied on such non-authenticated exchange and further verified the distributed key via subsequent 2-party mutual authentication handshake [7]; here we base the entire system on more novel authenticated key exchanges while not compromising the minimality achieved earlier.

In summary, 2PKDP lacks two features: i) integrity of the new key and ii) timeliness of the KDC's response. In order to address this issue, we construct a protocol which provides both features and, at the same time, retains the advantages of the basic protocol, most importantly, its compactness.



$$\boxed{MAC_a = MAC_a(N_a, N_k, KDC) \qquad N_k = K_a^{new}}$$

Figure 3: 2-Party Authenticated Key Distribution Protocol (2PAKDP)

The resulting protocol (2PAKDP) is illustrated in Figure 3. Its format is very similar to 2PAP in Figure 1. In fact, the only difference is that the KDC's nonce N_k in flow 2 (which also happens to be the new key K_a^{new}) is hidden. The technique used to construct message flow 2 is called *braiding* [20]. Upon receiving the message, A first performs block encryption on the MAC value and factors out the new key N_k. Then, it computes the MAC – as in the basic 2PAP – over (N_a, N_k, KDC) and checks that it matches the KDC's MAC in the message. The third protocol flow is identical to 2PAP and is, once again, optional.

The security of 2PAKDP is based directly on the security of the more basic 2PAP. Moreover, it is not difficult to see that 2PAKDP is, essentially, a superset of 2PAP. The arguments for supporting these assertions as well as the detailed derivation of 2PAKDP can be found in [20].

A remark on 2PAKDP is that, like its ancestors 2PKDP and 2PAP, it is a minimal protocol. In terms of composite message length, it is not possible for a principal to supply a challenge to the KDC and get back a new, unpredictable key and its integrity check in less than two messages and three data fields. Also, 2PAKDP requires only *one* additional (on top of 2PAP) block encryption operation in order to hide the key.

Finally, we remark that the protocol can be combined with a hard-to-compute function like Diffie-Hellman key exchange [12], to achieve better security (i.e., breaking the shared key does not give away the distributed key as in [2]). For example N_a and N_k may be the (authenticated) exchange messages of the key-exchange protocol (assuming the MAC is large enough, as for this technology, a larger key size is required).

3.5 Protocol Scenarios

We now proceed with the design of 3-party protocols. As mentioned before, two of our main guiding principles are i) protocol minimality, and ii) flexibility with respect to network configurations and connectivity patterns. We present optimized design that achieves authenticated key distribution in a minimal fashion (improving on the previous unauthenticated design in [7] where it was implicitly assumed that authentication is followed by key distribution).

The common setting for all protocol scenarios discussed below is exemplified by two entities A and B that want to authenticate with each other and, perhaps, subsequently engage in secure communication. Since initially A and B have no shared secret, they (or, rather, one of them) must contact the mutually-trusted KDC with whom they each share a secret.

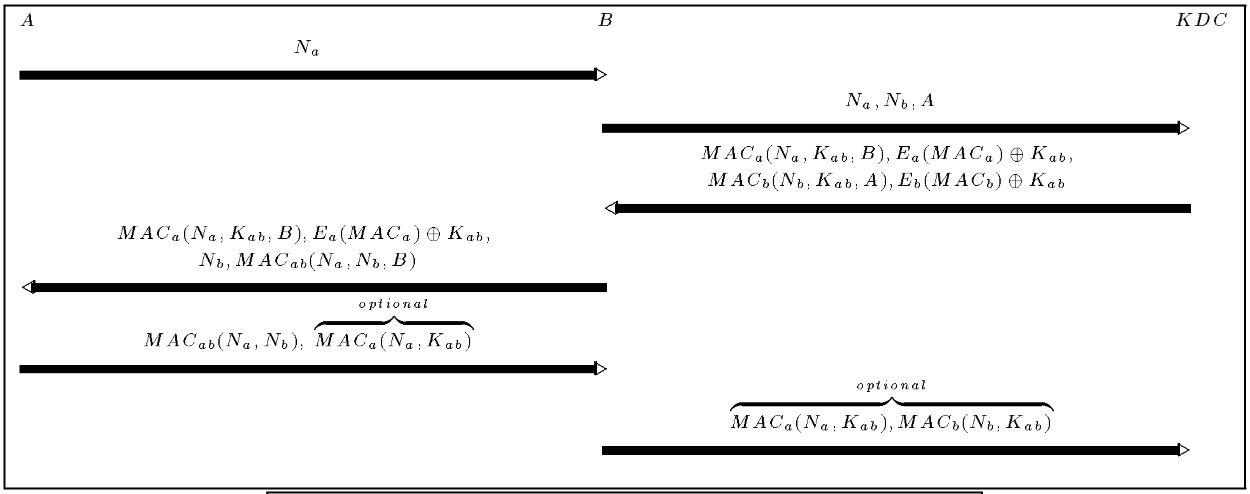
3.5.1 A-B-K Pull Scenario

The first scenario is called A-B-K or *pull model*. The A-B-K scenario assumes that A is either unable, unauthorized, or unwilling to contact the KDC or it is simply willing to let B choose if it wants to contact the KDC.

Our realization of A-B-K is shown in figure 4. It is constructed by simply combining and interleaving two 2PAKDP protocols ($A \leftrightarrow KDC$ and $B \leftrightarrow KDC$) and one 2PAP protocol ($A \leftrightarrow B$).

In more detail, the protocol is executed as follows:

1. As in the basic 2PAP, A contacts B in flow 1 challenging it to authenticate based on a nonce N_a.
2. B contacts the KDC in flow 2 passing along A's identifier, A's nonce N_a and its own nonce N_b. (We assume that B's name can be inferred from the underlying communication context.)



$$\boxed{MAC_a = MAC_a(N_a, K_{ab}, B) \qquad MAC_b = MAC_b(N_b, K_{ab}, A)}$$

Figure 4: A-B-K 3-Party Key Distribution Protocol

3. KDC replies in flow 3 by sending two tickets which contain the same freshly-generated key Kab. The tickets (or key-distribution expressions) follow the same syntax as in 2PAKDP.

4. Having received the tickets, B may or may not cache them for reuse in later communication with A. (The issue of caching tickets is discussed in a later section.) B extracts K_{ab} and checks its integrity and freshness by re-computing the MAC expression.

5. In flow 4, B forwards to A the appropriate ticket and adds its own nonce along with the expected authentication expression $MAC_{ab}(N_a, N_b, B)$.

6. In a similar fashion, A extracts and verifies K_{ab}, verifies the authentication expression from B, and replies with its own $MAC_{ab}(N_a, N_b)$. Optionally, A can add a confirmation token $MAC_a(N_a, K_{ab})$ that completes mutual authentication between A and the KDC.

7. Upon receipt of flow 5, B authenticates A's token thus completing the $A \leftrightarrow B$ 2PAP. Next, B can optionally compute its own confirmation, $MAC_b(N_b, K_{ab})$, and forward it along with A's confirmation to the KDC in the last flow.

The purpose of the optional confirmation flows is to acknowledge to the KDC that both A and B received the intended key K_{ab} and successfully authenticated one another. Such information may be useful for logging and subsequent auditing. Of course, if auditing is desired, the KDC needs to either maintain or archive state information associated with each protocol run.

This completes the description of the basic 3-party protocol. Whether A and/or B should cache its own and/or each other's tickets depends on the operational assumptions of the scenario at hand. If we assume that keys like K_{ab} are valid for one single interaction (message exchange or session duration) between A and B, then tickets need not be cached and keys should be stored and later erased at session termination time. This would force A and B to get a new key for each interaction. On the other hand, if the ticket usage conditions (e.g., validity interval) 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 K_{ab} if and when they resume interaction after a period of inactivity and within the lifetime K_{ab}.

3.5.2 K-A-B Push Scenario

The second scenario is called K-A-B or *push model*. In it, the KDC is contacted by A instead of B, as shown in figure 5. The KDC is contacted by A because, for some reason, B is unable, unwilling or unauthorized to do it.

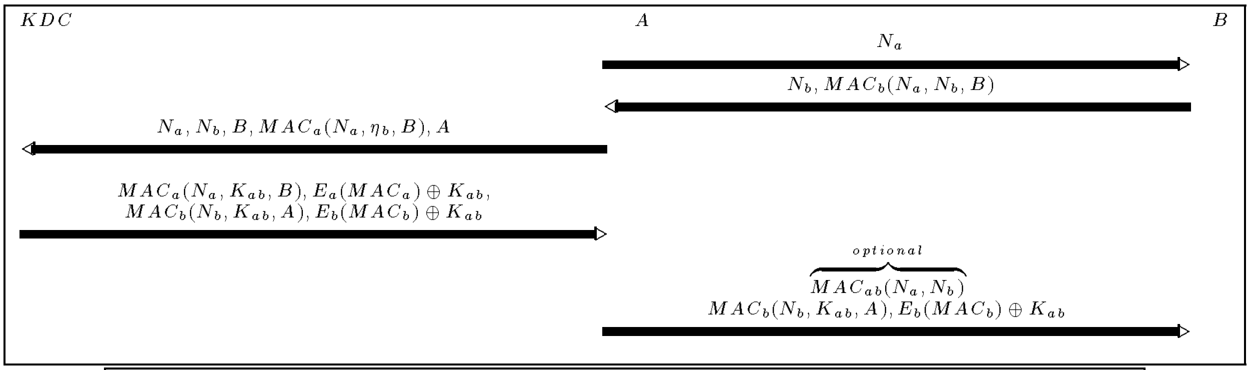
One distinguishing feature of this protocol is that it provides authentication on the forward flows to the KDC. By issuing the tickets in flow 4, the KDC confirms to both A and B that their respective authentication tokens – $MAC_b(N_a, N_b, B)$ and $MAC_a(N_a, \eta_b, A)$ – are valid. However, if either of the tokens is corrupted (B's in flow 2 or A's in flow 3), the KDC is unable to tell which one is at fault. In other words, the KDC can confirm that both tokens are valid or none are. In case it is desirable to pinpoint the corrupt token, flow 3 can be extended to include $MAC_b(N_a, N_b, B)$.

The most important consequence of forward-flow authentication is that the direct *handshake* (2PAP) between A and B is no longer needed (since it already takes place indirectly, through the KDC.) We note that the KDC only checks the authenticity of A's and B's tokens, but it does not and **cannot** assert the freshness of either token. The latter is inferred by A and B individually in flows 4 and 5, respectively.

We also note that $MAC_{ab}(N_a, N_b)$ in the last protocol flow is not always required. If we can guarantee no *crosstalk* between different 3-party protocols (as described in Section 3.6 below) then the MAC_{ab} expression can be safely omitted.[3]

Another distinguishing feature of K-A-B is that it does not provide the same assurances as A-B-K. In K-A-B, both principals are satisfied at the end that their peer is authentic and is currently on-line. In K-A-B, on the other hand, both principals gain an additional assurance that their peer is currently in possession of the new key K_{ab}. This difference becomes moot when the two principals engage in subsequent

[3] We leave it as an exercise to the interested reader to see why $MAC_{ab}(N_a, N_b)$ is needed otherwise.



$\eta_b = MAC_b(N_a, N_b, B) \qquad MAC_a = MAC_a(N_a, K_{ab}, B) \qquad MAC_b = MAC_b(N_b, K_{ab}, A)$

Figure 5: K-A-B 3-Party Key Distribution Protocol

communication secured with K_{ab}. However, since this is not always the case, we present a slight variation of K-A-B where $A \leftrightarrow B$ authentication is direct (see Figure 6.) The present protocol includes an extra flow that is required to complete the handshake. It is easy to see the resemblance to the A-B-K protocol discussed in the previous section. In fact, K-A-B* can be obtained by simply re-arranging the flows and message contents of A-B-K.

3.5.3 Time-Based Push Scenario

Under certain conditions, it may be desirable to use another flavor of the *push protocol*, referred to as K-A-B(t) and represented in figure 7. This protocol forces A to get the tickets from the KDC before communicating with B. There can be several reason for choosing K-A-B(t):

1. The main reason may be that A (or A's KDC) has to adhere to a policy whereby all requests have to be processed/approved by the KDC before any contact can be made. This is an issue which often depends on whether Mandatory Access Control is being enforced (connection between parties is strictly controlled by servers), in highly sensitive communication systems (e.g., military systems) such a policy is typically enforced.

2. B is not available (i.e., is not on-line) at the time when A contacts the KDC. Also, B may be a busy/popular service which cannot be burdened with maintaining state. (We note that both K-A-B and A-B-K require B to maintain state – N_b, A's name, etc. – from the time of initial contact by A until the ticket from the KDC is received.) In contrast, K-A-B(t) requires B to start keeping state starting only from flow 3.

3. B has no connection to, or is otherwise unable to contact the KDC. For example, B is a mobile entity that can only connect to the rest of the world through A. Of course, in this case K-A-B can be used as well but K-A-B(t) requires one message less than K-A-B.

Although the message structure of K-A-B(t) is quite different, it is not hard to see its resemblance to the Kerberos model. As the Kerberos protocol, K-A-B(t) assumes loose clock synchronization between B and the KDC. (however, no clock synchronization between A and KDC or A and B is assumed.[4])

[4] In practice, clocks of all principals involved must anyway be loosely synchronized in order to interpret the key expiration times that may be included as part of KDC-issued tickets.

The main distinguishing factor of K-A-B(t) is the lack of a challenge from B to the KDC. Instead, the timeliness/freshness of the KDC's ticket for B is established on the basis of the KDC-supplied time-stamp, TS_k. The protocol, as depicted, is a **hybrid** in a sense that $KDC \rightarrow A$ authentication is challenge-based as is $A \leftrightarrow B$ authentication while $KDC \rightarrow B$ is time-based. If tight clock synchronization between A and B is available, the protocol can be simplified to resemble even more the Kerberos model. In particular, instead of waiting for B's challenge, A can authenticate to B in flow 3 by including $MAC_{ab}(N_a, TS, A)$. Flow 4 can then be reduced to just $MAC_{ab}(N_a, TS)$ and the last flow can be omitted altogether. (See Figure 8.)

3.6 Protocol Coexistence

In order to provide seamless and flexible service, all protocol types described above have to coexist under a common *umbrella*, i.e., within the same body of code. This raises some concerns regarding an intruder being able to take advantage of any cross-talk or interplay between different 3-party protocols.

One intuitive argument against this type of attack is that only *quality ingredients* were used in the construction of the 3-party protocols. In other words, every 3-party protocol combines – in one way or another – two parallel runs of 2PAKDP and one run of 2PAP (and these components have been shown to withstand attacks of this kind). Thus, we argue that the security of 3-party protocols is based on the security of the underlying 2-party protocols. A rigorous proof of this assertion is outside the scope of this paper. It is based on (formally) having each flow of each protocol instance being independent of other cryptographic expressions to avoid reusing them either on- or off-line. One very effective measure for avoiding many types of attacks is the assignment of a unique identifier and include sender and receiver names in every cryptographic expression (MAC or E operation) in supported protocols, and the inclusion of this identifier in the computation of all such cryptographic expressions. This creates independence between cryptographic tokens of different protocols, and then we have also to pay attention to independence between instances of the same protocol. (In practice, this can be realized by appending or pre-pending the expression identifier to the cleartext before computing the MAC or E function over it. Alternatively, the key itself can be varied for every distinct cryptographic operation.) The formal argument can then be based on complexity theory (reducing breaking to breaking assumptions about the strength of the underlying tools) and/or be formalized in a

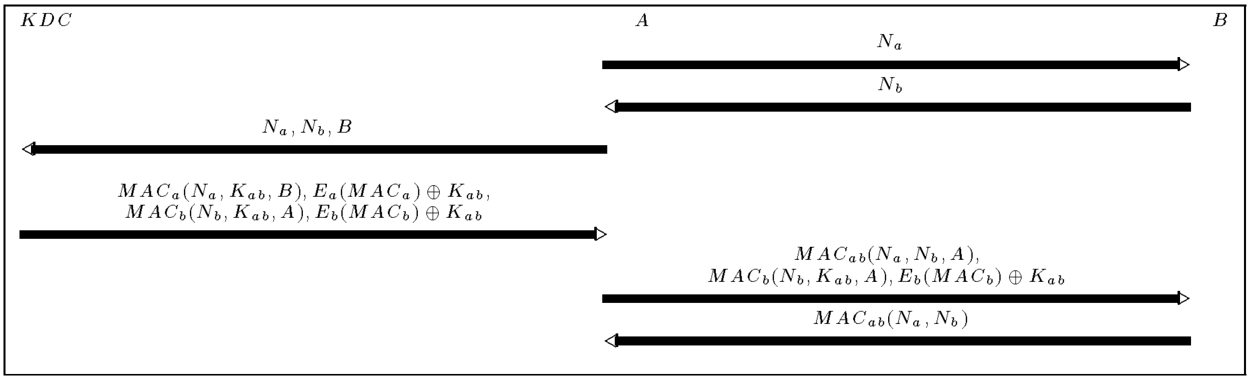


Figure 6: K-A-B* With Direct Handshake

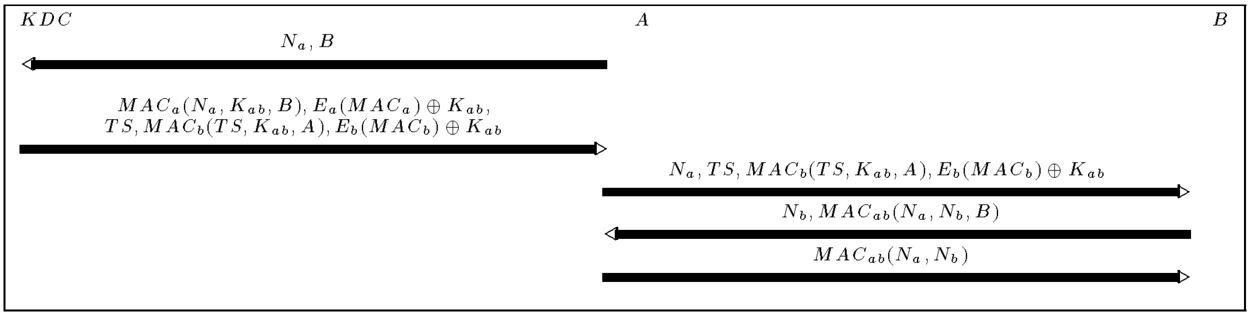


Figure 7: K-A-B(t) With Partial Clock Synchronization

logic that argues about properties of such protocols. Elegant formal complexity arguments have recently been developed based on the above intuition experienced in the project, see [3, 4].

Other aspects of protocol coexistence are the criteria for selecting a particular protocol (which depends on the environment definition, such as whether Mandatory Access Control is required).

4 Inter-Domain Protocols

There are many well-known reasons for partitioning electronic space into so-called realms or domains: scale, geographical distance and administrative heterogeneity, as well as security considerations are the most common of them. The growth of the Internet and its potential to be the universal host of many domains is one typical example. Whether it takes place across a private, relatively secure network or across the uncharted wilderness of the global Internet, communication increasingly crosses such *domain boundaries*. This prompts the need for security services to take into account the "inter-domain-ness" of communication. More specifically, authentication and key distribution protocols must be extended to operate across domains.

This requirement is certainly not new; for example, both MIT and DCE Kerberos breeds include an inter-domain key distribution protocol. (See Figure 9.) The Kerberos model puts most of the load on the initiating client A which is reasonable in the context of a traditional, LAN communication environment. However, the Kerberos model is not applicable to many other environments:

- Client A may be a mobile unit in which case it has no connection to its own KDC, least of all to a "foreign" KDC.
- Client A might have a means to reach its own KDC but not a foreign one. In fact, KDCs can migrate from one host to another, crash and come up again. Ideally, this should not be of any concern to anyone outside their own domain.
- Client A may have a policy prohibiting it from communicating with the outside without establishing a secure "context". This would be the case if packet filtering is being applied at domain boundaries. We note that recently there is a growing popularity to employ such filtering by means of "firewall" bridges.
- Client B may have a policy requiring it to consult its own KDC first before contacting A's KDC. (One example is a mandatory access control environment where external connections must be strictly monitored by KDCs.)

Keeping in mind the above conditions and the goal to design flexible protocols, we now extend our 3-party protocols to support inter-domain communication. Like Kerberos designers, we assume the existence of inter-domain keys – keys that secure communication among KDCs in different domains. An inter-domain key, K_{12} shared by KDC_1 and KDC_2 is assumed to be a strong, long-term key. In the descriptions below we assume symmetric, shared-key encryption, however, there is no reason why public key encryption could not be used for inter-KDC authentication and key distribution. The use of public key encryption can be made *visible* only to the KDCs and transparent to the regular principals. It would, however, imply longer messages and packet headers; thus, it is better suited to the versions with direct KDC-to-KDC communication.

Implicit in the above is the assumption that KDCs in different domains know one another *by name*. In other words, a KDC in every domain has a well-known (perhaps, reserved) name. Thus it should be easy to construct a name of a KDC given the name of its domain. However, we do not address

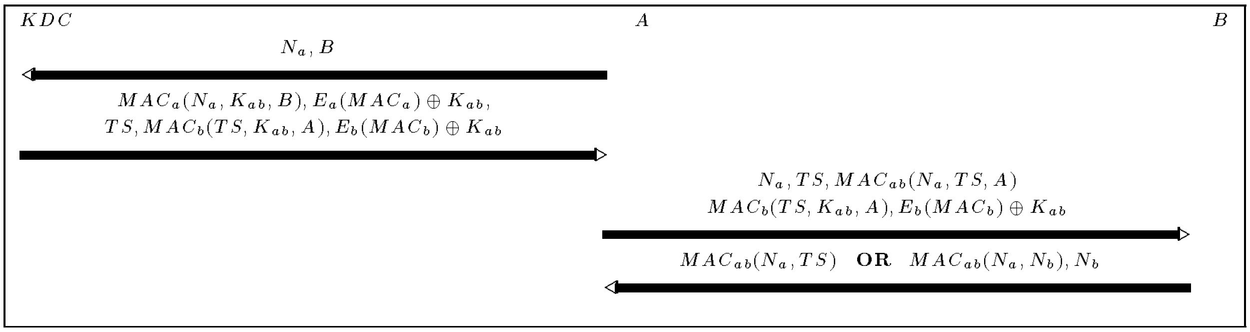


Figure 8: Fully Time-Based K-A-B(t)

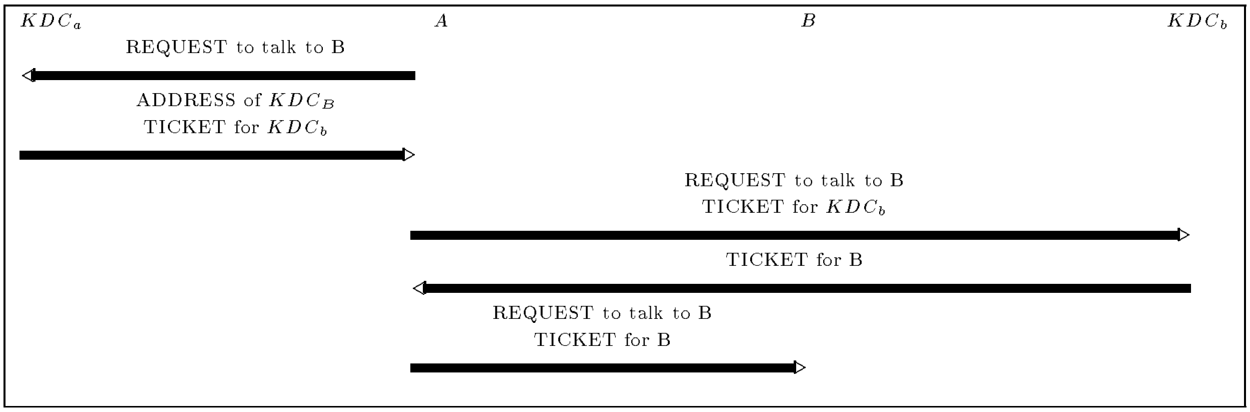


Figure 9: Kerberos Inter-Domain Key Distribution Model

any mechanisms for establishing relationships and distributing/refreshing keys between KDCs in different domains.

4.1 Without Inter-KDC Communication

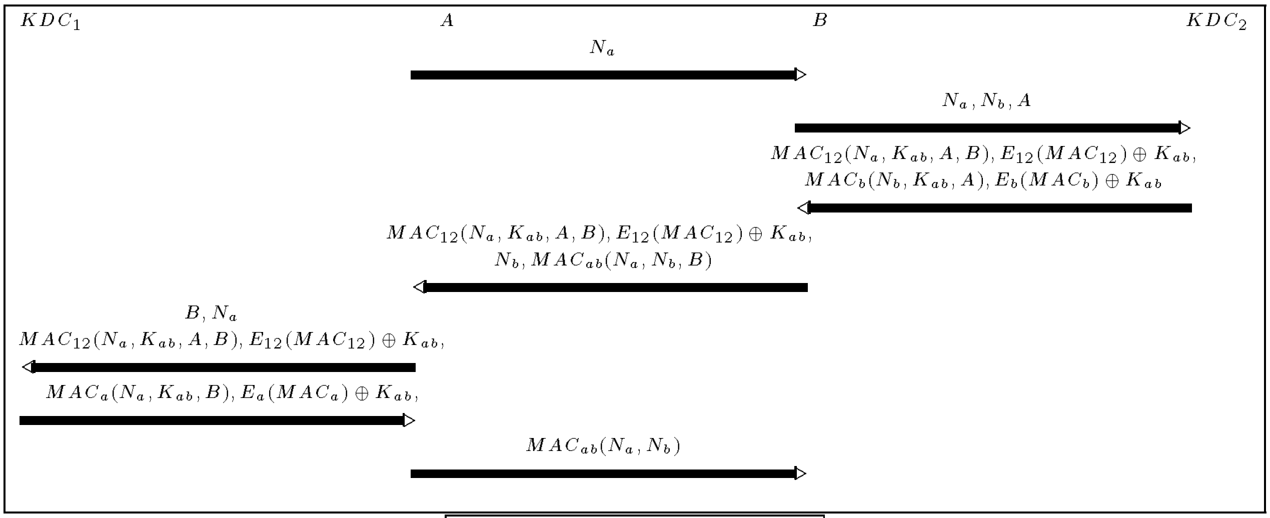
In the simplest inter-domain model we assume that KDCs do not engage in inter-domain communication. This restriction is realistic since the KDC is a special, perhaps protected entity and communication with the outside may result in its undue exposure. Furthermore, inter-domain communication is often less predictable and less reliable than intra-domain communication. Thus, the burden on the KDC can be reduced by disallowing outside communication. Finally, as mentioned before, KDCs can crash and migrate from host to host. Keeping track of the availability and the whereabouts of the KDCs is certainly more viable within a domain (where the KDCs are located) rather than outside.

Since direct communication between KDCs is unavailable, the obvious solution is for one of the KDCs to issue both tickets and for the other KDC to translate the appropriate ticket for its constituent principal. The protocol in Figure 10 depicts an example of this technique. It is essentially a straight-forward extension of the A-B-K protocol discussed in Section 3.5.1.

The protocol is executed as follows:

1. The first two flows are the same as in the intra-domain A-B-K protocol.
2. After receiving the request in flow 2, KDC_2 in B's domain discovers that one of the principals (A) is not local, i.e., not registered in the local database.
   (a) KDC_2 extracts A's domain name from A's name.[5]
   (b) From the domain name, KDC_2 determines the name of the peer KDC in that domain (KDC_1) and obtains the key (K_{12}) that it shares with that KDC. (The key can be stored, for example, in a normal principal entry in the local database of KDC_2.)
   (c) As in normal A-B-K, KDC_2 generates a new key K_{ab} and issues a ticket for B. However, since it does not know A's KEK K_a it cannot issue a ticket for A directly. Thus it issues a ticket for KDC_1 instead and forwards the reply to B.
3. B reads its own ticket and, behaving as usual, forwards A's ticket along with its own authentication token to A. We note that, throughout the protocol, B remains **oblivious** to the fact that A is in a different domain.
4. When A receives a ticket from B, it may do one of two things: i) try to read it (i.e., try to extract the key with K_a) or ii) simply determine from B's name that B is in a different domain and, therefore, the ticket coming from B is not readable. In any case, A forwards the ticket to KDC_1 for translation.
5. KDC_1 verifies the ticket and extracts K_{ab}. In doing so, it confirms that the ticket has been issued by KDC_2 and it authorizes A to "talk" to B. (The latter is confirmed by including both names in the computation of MAC_{12}.) However, KDC_1 can not draw any conclusions about the timeliness/freshness of the ticket it is translating. We do not consider this problematic since expiration time is expected to be factored into the ticket computation (although the protocol figures

[5] We assume that a principal name is always fully qualified, i.e., it includes as one of its components the domain name, e.g., $userX@domainY$.



$$\boxed{MAC_{12} = MAC_{12}(N_a, K_{ab}, A, B)}$$

Figure 10: A-B-K Inter-Domain Protocol

omit it.) In this sense, as long as the ticket is not expired, the KDC is willing to translate it.

This yields an unexpected benefit. Suppose that B caches A's ticket (as issued by KDC_2) and A does not cache its own ticket. At some later time B can supply A's ticket and A can get the ticket translated by simply asking KDC_1. Consequently, the two-message ticket translation sub-protocol (flows 5 and 6) can be thought of as an independent protocol in its own right.

6. The final flow from A to B is the same as in intra-domain A-B-K.

Other 3-party protocols can be extended in a similar fashion. The inter-domain version of the time-based K-A-B(t) protocol is shown in Figure 11. The main difference with A-B-K is that now KDC_1 issues the tickets and KDC_2 acts as a translator for B.

An inter-domain version of K-A-B is also possible but its construction is somewhat less straight-forward. (See Figure 12.) As discussed in Section 3.5.2, the distinguishing feature of K-A-B is the so-called *pre-authentication* on forward flows whereby the KDC authenticates both principals before issuing the tickets. This is no longer possible because, while KDC_1 can still authenticate A, it is unable to check B's $MAC_b(N_a, N_b, B)$ expression. The solution is to let KDC_1 issue the tickets *provisionally* by including B's MAC expression in B's ticket (which is, of course, not readable by B.) This is a slight compromise since in the original K-A-B both parties obtain their respective tickets only after both are authenticated.

We can avoid the compromise by constructing a rather *ugly* protocol where KDC_1 super-encrypts A's ticket with K_{12}. Then, A would have to forward its own ticket along with B's to KDC_2 (via B.) KDC_2 would, in turn, check B's MAC expression and only then "unscramble" A's and translate B's ticket. In addition to having an extra flow (from B to A to deliver the ticket), this protocol would require longer messages (in flows 5, 6 and 7.)

4.2 With Inter-KDC Communication

Although inter-KDC communication should be avoided at times, under certain circumstances it becomes a necessity:

1. One of the principals (A or B) has no connection to its own KDC. More generally, a principal may have no means of contacting a KDC be it its own or a foreign one.

2. Software (code) size at one or both principals is critical. In this case, it is necessary to hide the *inter-domain-ness* of the protocols from both principals. In other words, principals can pretend that their "world" consists of a single domain while the burden and the complexity due to inter-domain issues can be isolated in KDCs.

3. KDCs have separate (perhaps faster and/or more secure) communication facilities, i.e., KDCs can communicate among themselves more efficiently than principals. (This is precisely the case with many wireless/cellular networks.)

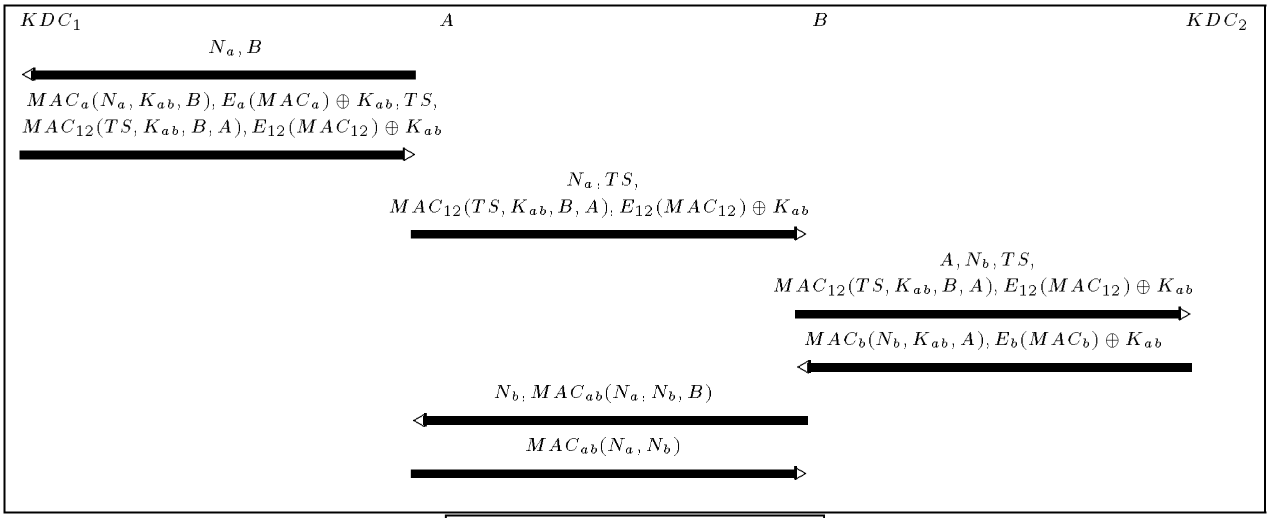
As can be expected, allowing inter-KDC communication is commensurate with additional protocol complexity. In particular, a KDC not only has to "know" its peer KDC(s) in a foreign domain (and the corresponding key), but it also has to be aware of their whereabouts and operational status.

4.3 Inter-KDC Communication

The actual protocol modifications needed to support inter-KDC communication are fairly trivial. Three resultant protocols: A-B-K-K, K-K-A-B and K-K-A-B(t) are shown in Figures 13-15 below. It is easy to see that each of the three protocols differs from its *intra*-domain counterpart only insofar as two additional flows between KDCs. As mentioned above, the protocols are transparent to the principals, e.g., A and B do not need to differentiate between, say, A-B-K and A-B-K-K.

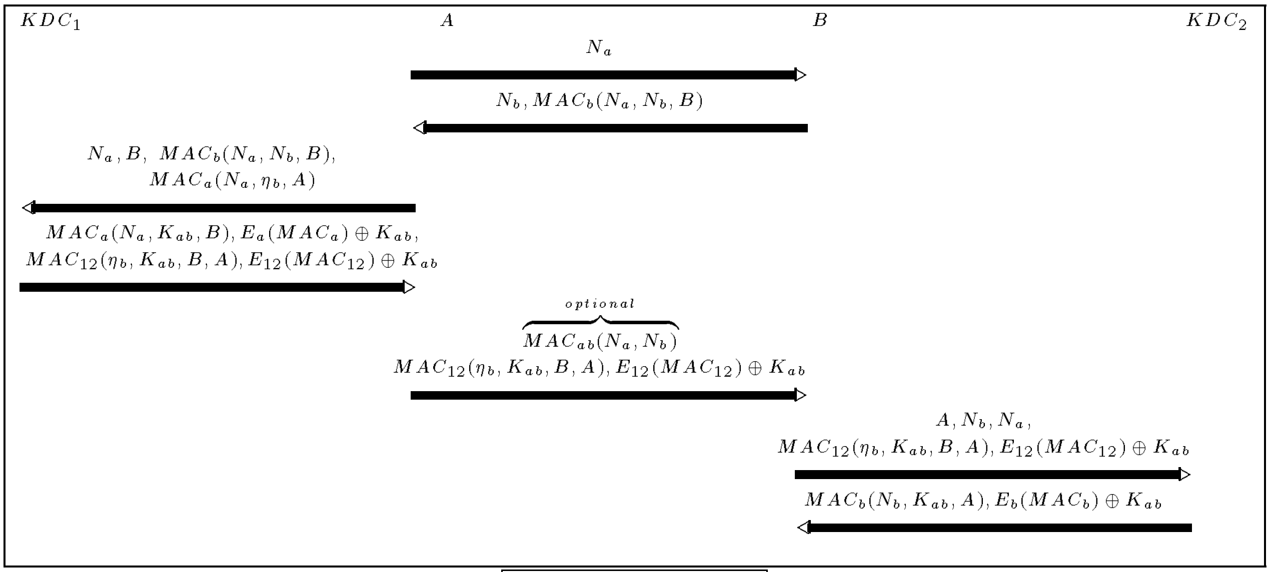
The main consequence of this convenience is the sacrifice of the KDC's *stateless* nature. Recall that all 2-party, 3-party and inter-domain protocols discussed so far allow KDCs to operate without having to keep state.[6] In other words, a KDC receives a request and replies to it; it does not wait for any external event or keep any state with respect to a request. A KDC never initiates communication; it only

[6] Excepting the password change protocol.



$\boxed{MAC_{12} = MAC_{12}(N_a, K_{ab}, B, A)}$

Figure 11: K-A-B(t) Inter-Domain Protocol



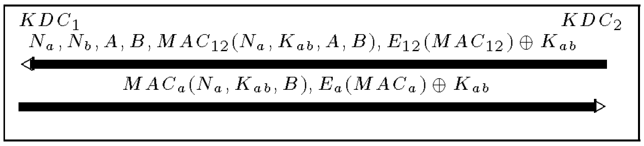
$\boxed{\eta_b = MAC_b(N_a, N_b, B)}$

Figure 12: K-A-B Inter-Domain Protocol

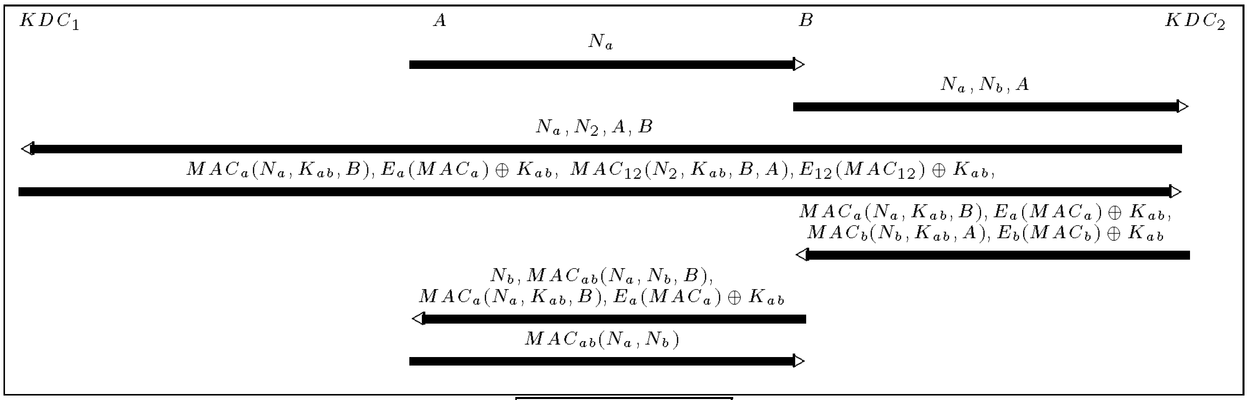
responds. All this keeps the design and operation of a KDC relatively simple. Inter-KDC communication changes the simple nature of a KDC since it is now required to initiate communication with peer KDCs, wait for the response and deliver the results to the requesting principal. The ensuing complexity and ways of dealing with it pose a number of interesting systems questions which, due to space limitations, we discuss only briefly in this paper.

4.3.1 A-B-K-K Protocol

As the contents of inter-KDC flows in Figure 13 indicate, KDC_1 issues the tickets (containing the new key) while KDC_2 only translates the ticket for B. Alternatively, the roles of the two KDCs can be switched as follows:



This variation would result in smaller messages but would also complicate things for KDC_2. This is because KDC_2 now has a choice: either it pre-issues a ticket for B before contacting KDC_1 (in which case it has to store the ticket in the meantime) or it has to issue the ticket for B when KDC_1 replies. In the latter case, KDC_2 still has to keep state between flows (at least A, B, K_{ab} and N_a must be kept.) Alternatively, KDC_2 can "export" the state information that it otherwise has to keep. This can be done by including the state variables in the flow to KDC_1; KDC_1 then simply echoes them back in its reply. (Of course, state variables can not be exported unprotected; K_{ab} must be encrypted and the rest must be at least integrity-protected.) This would, once again, involve longer and more compli-



N_2 – nonce of KDC_2

Figure 13: A-B-K-K Inter-Domain Protocol

cated messages. It becomes clear that the basic trade-off is between state retention (as well as implementation complexity) and message sizes.

Similar issues and trade-offs arise in the other two protocols: K-K-A-B and K-K-A-B(t).[7]

4.4 Other Protocols

In addition to the protocols described above, a complete protocol family must contain auxiliary protocols targeted at actual human users.

- *Login or Single Sign-On (SSO)*: protocol that authenticates the human user and distributes a session key. In its minimal form (user only has a weak password) it is susceptible to so-called dictionary attacks unless stronger techniques (e.g., encrypted-key-exchange [2], public-key encryption, or smartcards/tokencards) are used in conjunction. The *KryptoKnight* SSO protocol is discussed in [22].
- *Single Sign-On for Mobile Users*: protocol that handles the case of a mobile user attempting to log in from a foreign domain. In spite of the operational scenario being similar to the A-B-K protocol, mobile SSO has some rather esoteric requirements (see [23] for a complete discussion of this topic and *KryptoKnight* mobile SSO protocol.)
- *Password Change*: protocol for changing users' long-term passwords and keys. Like mobile SSO, this protocol has some special requirements (mainly in terms of fault-tolerance) that influence protocol construction. An in-depth treatment of the subject and the description of the *KryptoKnight* password change protocol can be found in [16].

5 Summary

The protocols presented in this paper form a comprehensive family that addresses most authentication and key distribution requirements for any network environment. We circumvent some limitations of existing designs by supporting a flexible range of network configurations, avoiding reliance on tightly-synchronized clocks or counters, minimizing message sizes and cryptographic operations, assuring scalability, and staying clear of export restrictions. Overall, the protocols are light-weight in terms of resource usage and suitable for low-end devices with limited memory and processing capability.

[7] One interesting aspect of K-K-A-B(t) protocol is that, unlike its immediate ancestor K-A-B(t), it does not require clock synchronization between the ticket-issuing KDC (KDC_1) and B. Instead, B's clock only needs to be synchronized with its own KDC_2. This is because KDC_2 has the option of updating the time-stamp (from TS_1 to TS_2) when translating B's ticket.

References

[1] R. K. Bauer, T. A. Berson, R. J. Feiertag, "A key distribution protocol using event markers," ACM TOCS 1 3 (1983) 249-255.

[2] S. Bellovin and M. Merritt, *Augmented Encrypted Key Exchange: a password-based protocol secure against dictionary attacks and password file compromise*, ACM Symp. on Comp. and Comm. Sec., 1993, 244–250.

[3] M. Bellare, P. Rogaway, "Entity authentication and key distribution", Proc. Crypto'93, Advances in Cryptology, Springer Verlag.

[4] M. Bellare, P. Rogaway, Provably Secure Session Key Distribution- The Three Party Case, STOC'95.

[5] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, "Systematic design of 2-party authentication protocols," Proc. Crypto 91, Santa Barbara, CA (Aug.91) 44-61. Lecture Notes in Comp. Sc. 576, Springer Verlag (1991).

[6] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, "Systematic design of a family of attack-resistant authentication protocols," IEEE JSAC special issue on secure communications (1992).

[7] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, "The *KryptoKnight* Family of authentication and key distribution protocols", IEEE/ACM Trans. on Netw. (March 95).

[8] D. E. Denning, G. M. Sacco, "Time-stamps in key distribution systems", CACM 24 8 (Aug.81) 533-536.

[9] "Data Encryption Standard", FIPS 46, NBS (Jan.77).

[10] Research Issues in Ubiquitous Computing, The 13-th ACM Symp. on Principles of Distributed Computing (PODC), 1994, 2–8.

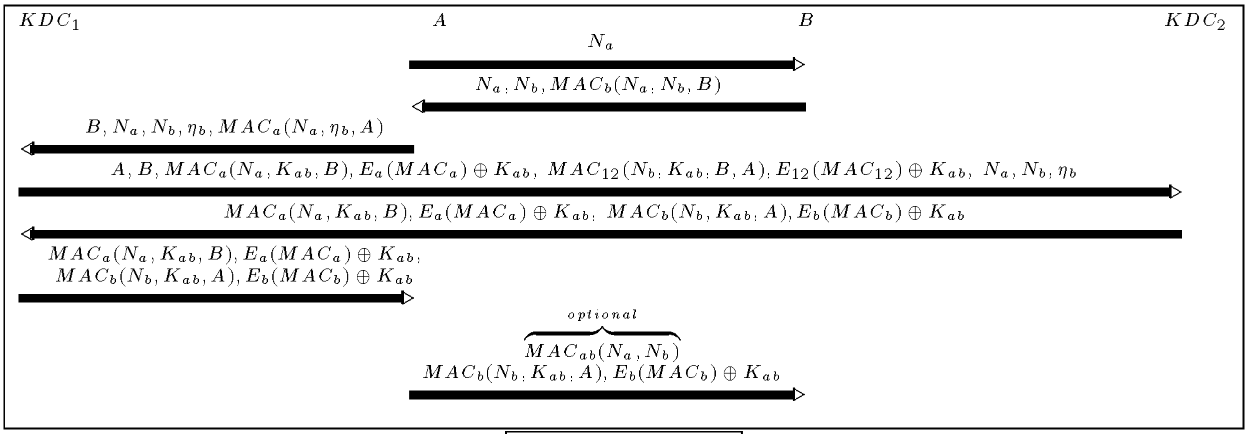
[11] W. Diffie, P. Van Oorschot, M. Weiner, *Authentication and Authenticated Key Exchange* Designs, Codes and Cryptography,2, 1992, 107-125.

[12] W. Diffie and M. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, November 1976.

[13] O. Goldreich, S. Goldwasser, S. Micali, *How to Construct Random Functions*, J. of the ACM, V. 33, 1986, 210-217.

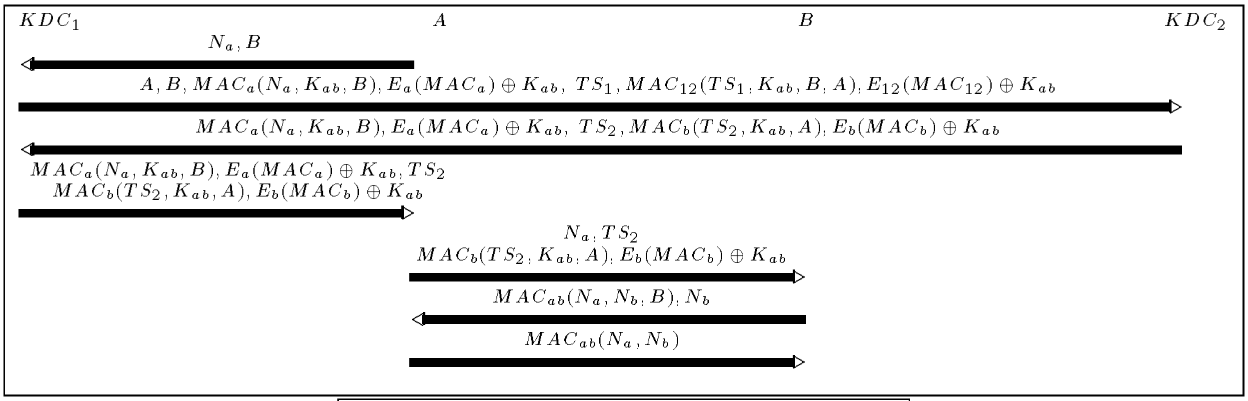
[14] L.Gong, *Using One-Way Functions for Authentication*, ACM Com. Comm. Review 19 5 (Oct.89) 8-11.

[15] L.Gong, Lower Bounds on Messages and Rounds for Network Authentication Protocols, The 1-st ACM Conf. on Compu. and Comm. Security, 1993, 26-37.



$\boxed{\eta_b = MAC_b(N_a, N_b, B)}$

Figure 14: K-K-A-B Inter-Domain Protocol



TS_1, TS_2 – time-stamps of KDC_1 and KDC_2, respectively

Figure 15: K-K-A-B(t) Inter-Domain Protocol

[16] R. Hauser, P. Janson, R. Molva, G. Tsudik and E. Van Herreweghen, "Robust and Secure Password/Key Change Method", 1994 European Symposium on Research in Computer Security, November 1994.

[17] ISO CD 9798-2, Information Technology: Security Techniques– Entity Authentication Mechanisms. Part 2: Entity Authentication Using Symmetric Techniques. July 1993.

[18] "Banking - Key management (wholesale)" ISO 8732, Geneva (1988).

[19] "OSI Directory - Part 8: Authentication Framework", ISO 9594-8, Geneva (1988).

[20] P. Janson and G. Tsudik, "Secure and Minimal Protocols for Authenticated Key Distribution", Computer Communications, September 1995.

[21] A. Kehne, J. Schoenwaelder, H. Langendoerfer, "A nonce-based protocol for multiple authentications", ACM OSR 26 4 (Oct.92) 84-89.

[22] R. Molva, G. Tsudik, E. Van Herreweghen, S. Zatti, "*KryptoKnight* authentication and key distribution system," Proc. ESORICS 92, Toulouse, France (23-25 Nov.92).

[23] R. Molva, D. Samfat and G. Tsudik, "Authentication of Mobile Users" IEEE Network, Special Issue on Mobile Communications, March/April 1994.

[24] M. Naor, M. Yung, "Universal One-Way Hash Functions and Their Cryptographic Applications," ACM Sym on Theo. of Computing, 1989.

[25] R. M. Needham, M. D. Schroeder, "Using encryption for authentication in large networks of computers," CACM 21 12 (1978) 993-998.

[26] R. M. Needham, M. D. Schroeder, "Authentication revisited", ACM OSR 21 1 (Jan.87) 7.

[27] B. C. Neuman, S. G. Stubblebine, "A note on the use of timestamps as nonces", ACM OSR 27 2 (Apr.93) 10-14.

[28] B. C. Neuman, T. Ts'O "Kerberos: An Authentication Service for Computer Networks", IEEE Communication Magazine, Sep. 1994.

[29] D. Otway, O. Rees, "Efficient and timely mutual authentication", ACM OSR 21 1 (Jan.87) 8-10.

[30] R. Rivest, "The MD5 Message Digest Algorithm", Internet draft, RSA Co. (Jul.91).

[31] J. Rompel. "One-Way Functions are Necessary and Sufficient for Digital Signature," ACM Sym. on Theo. of Computing, 1990.

[32] G. Steiner, C. Neuman, J. I. Schiller, "Kerberos: an authentication server for open network systems", Proc. Usenix Conf. (Winter 88).

[33] G. Tsudik, "Message authentication with one-way hash functions," Proc. IEEE Infocom 92, Florence, Italy (May 92).

[34] G. Tsudik, E. van Herreweghen, On Simple and Secure Key Distribution, 1-st ACM Conf. on Compu. and Comm. Security, 1993, 49-57.