On Simple and Secure Key Distribution

Gene Tsudik Els Van Herreweghen

Communications and Computer Science Department
IBM Zürich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{gts,evh}@zurich.ibm.com

Abstract

Many recent research efforts in computer security focus on constructing provably secure authentication protocols. Although many of the resulting protocols rely on the a priori *secure distribution of secret keys, no provably secure key distribution protocols have yet been demonstrated. In this paper, we use an existing secure two-party authentication protocol as a stepping stone for constructing a series of simple and secure key distribution protocols. The protocols are shown to satisfy desired security requirements, using the security properties of the underlying authentication protocol.*

1 Introduction

Research in authentication protocols has been fairly active since the publication in the late 1970s of the Needham and Schroeder landmark paper [11]. In it, they proposed a family of protocols for two- and three-party authentication and key distribution. The distinguishing feature of these protocols was the use of encryption for authentication. The protocols were subsequently shown to contain some subtle weaknesses (e.g., [5]). These weaknesses were subsequently fixed in [12] and the Needham-Schroeder protocol family served as a basis for the well-known Kerberos network security server originally developed at MIT [15, 16]. Although popular and fairly widespread, Kerberos has been subject to considerable criticism both for the protocols it is based on as well as for their implementation (see, for example, [1].)

More recently, research efforts began to develop (or to provide tools needed for developing) authentication protocols with some formal assurance of security. Most notably, formal tools for testing authentication protocols were developed, e.g., BAN logic [4] and its successor, GNY logic [8]. Instead of devising specific protocols, these methodologies provide a means for showing that appropriate beliefs are attained as a result of running an authentication protocol. They are, however, no panacea as there remain a number of issues (i.e., non-disclosure of secret information) that these methodologies do not address.

Other efforts, in particular, Bird et al. [2], concentrated on constructing protocols that can be shown secure (in some restricted definition of the term.) The main result of Bird et al. [2] is a two-party authentication protocol shown to be secure against an important class of attacks known as *interleaving* attacks. Such attacks are based on the adversary's ability to use either i) legitimate flows obtained from past executions of the protocol or ii) protocol flows elicited by the adversary from legitimate parties.

Both efforts stopped short of designing more elaborate protocols, specifically, key distribution protocols that can be shown secure. In fact, the follow-up to [2] is a family of three-party key distribution and authentication protocols [3] that has been subsequently realized in an actual network security service, KryptoKnight [9]. The underlying key distribution protocols, however, have not been shown secure; at least, not insofar as the authentication protocols in [2].

The goal of this paper is to probe further, i.e., to develop simple and secure two- and three-party key distribution protocols. This is achieved by combining careful protocol construction techniques and utilizing the properties of a two-party authentication protocol (already proven secure) as basic building blocks. Our design is similar both in spirit and motivation to techniques developed by Gong in [7], but focuses more on modularity and simplicity.

2 Design Goals

Before turning to the construction of the actual protocols, we emphasize the goals of our design and desired properties of the resulting protocols:

- Simplicity
  Simplicity is the major theme in our design and its foremost intended feature. The simpler the protocol, the easier it is to spot vulnerabilities and to demonstrate security features.

In Proceedings of ACM Conference on Computer and Communications Security, November 1993.

- No timestamps
  Use of timestamps in authentication and key distribution protocols has been debated *ad nauseum* for a number of years. Since our concern is with simplicity, timestamps are unacceptable because of the inherent requirement for (even loose) clock synchronization.

- Small number of cryptographic operations
  Cryptography is essential but must be used sparingly. Minimizing the use of cryptography makes protocols simpler and more efficient.

- Small message sizes
  Small message sizes can make a protocol suitable for implementation in space-conscious environments, e.g., in a network layer or in a boot service. Another incentive is to eliminate unnecessary redundancy which can otherwise make a protocol less secure and/or less efficient.

- Small number of messages
  Similarly, too many messages make an awkward protocol. Few messages make the protocol simpler to implement and less prone to timing constraints (i.e., fewer delays).

- Conventional cryptography
  The merits of public key cryptography are many and well-known. However, it is still quite inefficient. Furthermore its use sometimes presents a problem because of the associated patent issues. Finally, most public key methods impose a fairly large basic encryption block size (e.g., a 512 bits is a recommended minimum for RSA [14].) In light of these concerns we proceed with conventional encryption in mind. Nonetheless, for the sake of generality, the resulting protocols must not have any features that rule out the use of public key cryptography.

- No decryption
  A typical cryptosystem has two components: encryption and decryption. While the use of encryption is necessary there are reasons to avoid using decryption. First, decryption makes the implementation more complex. Second, it rules out the use of strong one-way hash functions in place of traditional encryption techniques (and where only encryption is needed, strong one-way hash functions can be used instead at much lower cost; see, for example, [17].) Finally, protocols using only encryption (but no decryption) avoid the entire issue of exportability which is relevant at least in the United States.

- Minimal overhead
  The key distribution protocol we intend to construct should impose minimal additional overhead on the existing authentication protocol it is based on. Since the authentication protocol used as building block is a generic one, we abstract out the specifics of the underlying encryption function.

3 Terminology

The following terminology is used throughout the paper:

- A, B, P, Q - full principal names (e.g., in X.500 format).

- S - full name of the Authentication Server; S is assumed to be universally trusted by all constituent principals.

- $\oplus$ - exclusive-OR operator.

- $E_K(X)$ - encryption of plaintext block X under key K. Both K and X are assumed to be not longer than the basic block size of the underlying encryption function, e.g., 64 bits in case of DES [10] or 512 bits in case of MD5 [13].

- $MAC_K(X)$ - DES-style message authentication code (MAC) computed in CBC mode over plaintext X with key K. However, this notation does not in any way imply the use of DES.

- K_{ab} - all symbols beginning with K are keys. The two-letter subscript identifies the two principals sharing that key (e.g., A and B share K_{ab}).

- N_{ab} - all symbols beginning with N are nonces, i.e., non-secret but unpredictable random numbers, used only once for this purpose. The first letter of the subscript (e.g., a in N_{ab}) refers to the party that originated the nonce while the second subscript letter identifies the target party.

- $A \Longrightarrow \mathrm{B}\ \ \mathrm{X}$ - this notation captures a single protocol flow; it denotes that A sent a message X to B.

4 Initial Assumptions

The single most important assumption we need to make is that *there exists a secure two-party authentication protocol (2PAP)*. One example is the nonce-based three-message protocol developed in [2]. The protocol is illustrated in Figure 1. However, **any** secure nonce-based 2PAP will suffice for our purposes. Informally speaking, a 2PAP is considered secure if and only if:

It is computationally difficult for an intruder to impersonate either party

The *difficulty* should be equal to the strength of the underlying cryptosystem or a strong one-way function. For example, if DES [10] is used with the 2PAP in [2], the computational difficulty of defeating the protocol equals that of breaking DES by brute force, which is generally believed to require on the order of 2^{56} trials.

$P \Longrightarrow$ Q	P, N_{pq}
$Q \Longrightarrow$ P	$AUTH_{K_{pq}}(N_{pq}, N_{qp}, Q), N_{qp}$
$P \Longrightarrow$ Q	$ACK_{K_{pq}}(N_{pq}, N_{qp}, P)$

Figure 1: Secure Two-Party Authentication Protocol (2PAP)

$AUTH_{K_{pq}}$ denotes an authentication expression based on the shared key K_{pq} and generated by the responding party (Q in our case). This expression is computed over three inputs: two nonces (one generated by each party) and the name of the message originator. One example of $AUTH$ is:[1]

$$E(Q \oplus E(N_{qp} \oplus E(N_{pq})))$$

Similarly, $ACK_{K_{pq}}$ is the authentication expression that the initiating party sends in order to complete two-way authentication. It is computed over the same inputs except for the message originator's name (which is P in this case, or can even be omitted as shown in [2]). An example of ACK is:

$$E(N_{qp} \oplus E(N_{pq}))$$

The above protocol has several advantages:

- Compactness: minimal number of messages; also, each message is of minimal length.
- Security: the protocol has been shown secure against interleaving attacks.
- No decryption: a strong one-way function suffices.

5 Two-party Key Distribution Protocol (2PKDP)

The model for two-party key distribution is such that one of the parties initiates the protocol by requesting a *new* key. The other party responds by generating a new key and shipping it back to the requester. The protocol may include a confirmation flow whereby the initiator acknowledges the receipt of the new key.

Although our ultimate goal is the design of a secure three-party key distribution protocol (3PKDP), we adopt an incremental approach by first constructing a secure two-party KDP and, subsequently using it as a stepping stone for obtaining three-party KDPs. However, one should not conclude that two-party key distribution, by itself, has no applications. A 2PKDP can be used, for example, to refresh a short-term session key between two parties (while retaining a more long-term pairwise key.) More generally, a 2PKDP can be used in an environment where the requesting party needs a good, fresh key for any number of reasons and, not being able to generate such a key, asks the party that can (e.g., an AS.)

5.1 Desired Properties

A 2PKDP is said to be secure if and only if the following properties hold:

- **Key Non-Disclosure**: a third party cannot discover a key being distributed without explicit collaboration of a legitimate party. (Legitimate party is one of the two protocol participants.)
- **Key Non-Modification**: a third party cannot modify a key being distributed to any value known to this third party.
- **Key Non-Reuse**: a third party cannot distribute a previously used key, i.e., it cannot fool the initiating party into using an old key. (This can be considered a subset of the Key Non-Modification property.)
- **Key Independence**: knowledge of one key cannot be used to compute other keys, i.e., a key distributed in one protocol run does not open the door to discovering keys distributed in other protocol runs.

Strictly speaking, the integrity of the key being distributed is not one of the goals of a "pure" KDP. In other words, an attacker may be able to tamper with a key distribution message, resulting in a different, albeit unknown to the attacker, value of key. This seems a bit unorthodox, however, it can be argued that, once a key is distributed, a simple authentication protocol between the two parties can be used to verify the "goodness" of the key.

[1] All encryption is performed with K_{pq}.

5.2 Constructing Secure 2PKDP with Secure 2PAP

In this section we derive some features of cryptographic expressions used in the secure 2PAP protocol. These will help us later in demonstrating desired properties of the key distribution protocols presented in subsequent sections.

Lemma 1: Given a secure 2PAP of the form depicted in Figure 1, it is computationally difficult for an intruder (not knowing K_{pq}) to obtain an $AUTH$ expression when at least one of the nonces N_{qp} or N_{pq} is selected by a legitimate party.

Proof: Follows directly from our definition of *secure* 2PAP.

Let $F(K, P) = C$ be a strong encryption function that transforms plaintext P into ciphertext C using key K, e.g., DES. Let $\bar{K}$ be a randomly-generated, unpredictable, used-only-once quantity, i.e., $\bar{K}$ is a *good nonce.*

Lemma 2: It is computationally difficult to obtain:

$$F(AUTH_{K_{pq}}(N_{pq}, N_{qp}, Q), \bar{K})$$

without the knowledge of the $AUTH$ expression.

Proof: Since the $AUTH$ expression itself is computationally difficult to obtain, it follows that any encryption of randomly-selected cleartext ($\bar{K}$) computed with $AUTH$ as a key, is also computationally difficult to obtain.

Corollary: It is computationally difficult to obtain:

$$AUTH_{K_{pq}}(N_{pq}, N_{qp}, Q) \oplus \bar{K} \tag{1}$$

Proof:

Expression 1 represents one-time-pad encryption ($\oplus$) of "plaintext" $\bar{K}$ with "key" $AUTH_{K_{pq}}(N_{pq}, N_{qp}, Q)$. Both plaintext and key are random and unpredictable values, and therefore this can be considered a strong encryption function.

5.3 Sample Protocol

$P \Longrightarrow$ Q	P, N_{pq}
$Q \Longrightarrow$ P	$AUTH_{K_{pq}}(N_{pq}, N_{qp}, Q) \oplus K_{new},\ N_{qp}$

Figure 2: Two-Party Key Distribution

We now turn to the particulars of 2PKDP. The protocol depicted in Figure 2 takes advantage of the $AUTH$ expression scavenged from the secure 2PAP. It is a simple protocol consisting of only two messages. It resembles a truncated 2PAP except that the authentication expression in the second message is used as a one-time mask for the key being distributed.[2] K_{new} is assumed to be a good nonce, i.e., random, used-only-once value. K_{pq} is assumed to be a strong long-term key (e.g., a key obtained at login time) while K_{new} is a short-term session key. In order to obtain K_{new} the recipient (P) first recomputes the mask based on its own N_{pq} and N_{qp} which is supplied in cleartext. The key is then extracted by XOR-ing the re-computed mask expression with the corresponding field in the message.

Key disclosure in 2PKDP can take place only if the attacker is able to obtain an $AUTH$ expression that *safeguards* a new key. The attacker has two sources of information that can help in "breaking" the protocol. First of all, the attacker may record any number of legitimate executions of 2PKDP between P and Q. In this case, N_{pq} is always under control of P and N_{qp} is always under control of Q. Alternatively, he may try to impersonate P by changing or composing a first message of the protocol and intercepting Q's reply; in which case N_{pq} is under the attacker's control, and N_{qp} is selected by Q. In both cases, at least one of the nonces: N_{pq}, N_{qp}, is **always** under the control of a legitimate party, i.e.,P or Q. Therefore, the ability to compute $AUTH_{K_{pq}}(N_{pq}, N_{qp}, Q)$ is equivalent to breaking 2PAP.

Key modification Key modification to a value known by the attacker is essentially equivalent to key disclosure. If the attacker is able to modify the key to a selected value then the corresponding $AUTH$ expression simultaneously becomes known. However, since the attacker cannot know the key *a priori*, he must first know the $AUTH$ expression. This is clearly impossible, as shown above.

Key re-use entails the attacker "feeding" an old key to the initiating party. Note that the attacker does not have to **know** the old key in order to try this attack (the simplest attack is to use pre-recorded replies from previous 2PKDP runs.) Since the attacker does not know any old key K^o, the only pieces of knowledge available to him are the recorded messages from previous protocol runs of the form:

$$AUTH_{K_{pq}}(N^o_{pq}, N^o_{qp}, Q) \oplus K^o,\ N^o_{qp} \tag{2}$$

In order to be fooled into accepting K^o, P has to receive a message of the form:

$$AUTH_{K_{pq}}(N_{pq}, N^x_{qp}, Q) \oplus K^o,\ N^x_{qp}$$

where N_{pq} is the fresh nonce generated by P in the current protocol run and N^x_{qp} is a nonce which is either

[2] It is assumed that the $AUTH$ expression yields a uniform-random expression of the same size as the key, which is the case for the protocol depicted in Figure 1.

generated by the attacker or by Q. Then, the following relationship must hold:

$$AUTH_{K_{pq}}(N_{pq}, N^x_{qp}, Q) \; = \; AUTH_{K_{pq}}(N^o_{pq}, N^o_{qp}, Q)$$

Assuming that $AUTH$ is based on a strong one-way function, this condition can hold only if:

$$N_{pq} = N^o_{pq}$$

and

$$N^x_{qp} = N^o_{qp}$$

which is impossible since P is assumed to generate *bona fide* nonces (i.e., nonces, by definition, are never reused.)

Alternatively, we can analyze the issue of *replay* by considering what happens if an attacker re-sends an old protocol message (see expression 2 above) to P, replacing the nonce N^o_{qp} by some value N^x_{qp}:

$$AUTH_{K_{pq}}(N^o_{pq}, N^o_{qp}, Q) \oplus K^o, \;\; N^x_{qp}$$

where either $N_{pq} \neq N^o_{pq}$ or $N^x_{qp} \neq N^o_{qp}$, P will extract a key K' instead of K^o, satisfying following equation:

$$AUTH_{K_{pq}}(N_{pq}, N^x_{qp}, Q) \oplus K' = AUTH_{K_{pq}}(N^o_{pq}, N^o_{qp}, Q) \oplus K^o$$

K' is, however, not known to the attacker, since he can't compute the AUTH expression masking it.

One important consequence of this result is that the attacker can, in effect, "fool" the protocol initiator P into accepting just about any tuple of the form $< G^x, N^x >$ as valid reply in the second flow of the protocol. However, we still claim that **the protocol achieves its goal** of distributing a random, one-time key which can only be discovered by the legitimate protocol initiator. That is, even though the attacker can convince P to accept any value G^x as an expression masking the key, the protocol retains its strength since the key extracted from G^x remains secret.

Key independence requires that protocol runs be unrelated. If the attacker discovers K^i from some protocol run (possibly via a brute force attack or some other means external to the protocol) then he simultaneously gains the knowledge of the corresponding $AUTH$ expression, $AUTH_{K_{pq}}(N^i_{pq}, N^i_{qp}, Q)$ that conceals the said key.

As there is no relationship between keys distributed in different protocol runs, knowledge of a K_i, by itself, offers no advantages to the attacker. Even knowledge of $AUTH_{K_{pq}}(N^i_{pq}, N^i_{qp}, Q)$ is only marginally useful since we assume that the probability of:

$$AUTH_{K_{pq}}(N^i_{pq}, N^i_{qp}, Q) = AUTH_{K_{pq}}(N^j_{pq}, N^j_{qp}, Q)$$

is negligible when $[N^i_{pq}, N^i_{qp}] \neq [N^j_{pq}, N^j_{qp}]$ (i, j denote the different protocol runs.) Therefore, knowledge of a single session key cannot lead to the discovery of other session keys.

Key Integrity The protocol in Figure 2 above does not provide for key integrity. As discussed before, key integrity is not necessarily considered a foremost property of a secure key distribution protocol. Failure to assure key integrity may result in the distribution to the requesting party of a key different from the one originally issued. However, under some circumstances, this is not problematic. For example, if the key issuer does not retain any knowledge of the key after issuing it (the key may be intended for communication between the requester and some other party), the integrity of the key does not matter so long as its value does not become known to an unauthorized party.

There are also scenarios where key integrity is needed. So far, we have made an implicit assumption that the new key is chosen **uniformly** by its issuer. By *uniformly* we mean that, supposing that a key is an n bits long, then *every* possible n-bit quantity is equally likely to be selected as a key. In this case, any modification of the key distribution token (in message 2) still preserves the uniformity of the key that is subsequently extracted. On the other hand, if keys are selected in a non-uniform manner whereby each key must satisfy some particular requirements (e.g., the RSA cryptosystem), the non-uniform properties would not be preserved if the key distribution token is modified.

6 Three-party Key Distribution Protocol (3PKDP)

The properties of the 2PKDP discussed so far appear reassuring. However, two-party key distribution is not a particularly useful application. A much more common scenario is that of three-party key distribution. The model for three-party key distribution is that two parties having no shared secret key enlist the assistance of a mutually-trusted third party performs the actual key distribution. This trusted third party is frequently referred to as Authentication Server (AS) or Key Distribution Center/Server (KDC). Each of the two parties are assumed to share a longer-term key with the AS.

6.1 Desired Properties

As with 2PKDP, the goal is to design a *secure* 3PKDP. The conditions for a secure 3PKDP are essentially similar to that of 2PKDP. The only additional requirement is that a 3PKDP must be secure against a malicious *insider*, i.e., a legitimate party that, by participating in legitimate runs of the protocol, can gather enough information to impersonate other parties or otherwise abuse the protocol.[3] This requirement is, of course, limited since there are some exposures (e.g., a malicious insider disclosing a key shared with another party) that cannot be addressed within the scope of a 3PKDP.

6.2 The Protocol

Figure 3 illustrates a *naive* version of a secure 3PKDP.[4] It is constructed by simply putting together two runs of 2PKDP.[5] One notable aspect is that the key being distributed in messages 2 and 4 is one and the same – K_{ab}. The names of the parties involved are changed to emphasize the difference with respect to previously discussed two-party protocols; A and B are the two principals and S is the mutually-trusted AS. The only other aspect where the present protocol differs from 2PKDP is in the way principal names are used within $AUTH$ tokens. Whereas before, a name denoted the originator of a token (as in Figure 2), it now refers to the thrid-party in the protocol, e.g., the $AUTH$ token sent from S to A includes B's name; similarly, the $AUTH$ token sent from S to B includes A's name. This feature is necessary to prevent masquerading attacks whereby a malicious party tampers with the principals' names in message 1 of the protocol.

$A \Longrightarrow$ S	A, B, N_{as}
$S \Longrightarrow$ A	$AUTH_{K_{as}}(N_{as}, N_{sa}, B) \oplus K_{ab},\ N_{sa}$
$B \Longrightarrow$ S	B, A, N_{bs}
$S \Longrightarrow$ B	$AUTH_{K_{bs}}(N_{bs}, N_{sb}, A) \oplus K_{ab},\ N_{sb}$

Figure 3: Naive Three-Party Key Distribution

The protocol is secure with respect to outsider attacks, i.e., a non-participating party (i.e., not A, B or S) cannot subvert the protocol. This follows directly from the established security of 2PKDP. The only additional information available to the attacker from 3PKDP (as opposed to two unrelated runs of 2PKDP) is the fact that the same key is being distributed to A and B. However, not knowing either the key or the masking expression, the attacker can only try to play XOR-ing "games" and factor out K_{ab} by computing:

$$AUTH_{K_{as}}(N_{as}, N_{sa}, B) \oplus K_{ab} \oplus AUTH_{K_{bs}}(N_{bs}, N_{sb}, A) \oplus K_{ab} = AUTH_{K_{as}}(N_{as}, N_{sa}, B) \oplus AUTH_{K_{bs}}(N_{bs}, N_{sb}, A)$$

This expression cannot be of any value since its components remain unknown. In fact, the attacker can succeed in distributing $AUTH_{K_{as}}(N_{as}, N_{sa}, B)$ to B (instead of K_{ab}) and/or $AUTH_{K_{bs}}(N_{bs}, N_{sb}, A)$ to A (instead of K_{ab}). However, this is nothing but a special case of previously discussed key integrity issue. Recall that even in 2PKDP the attacker is able to modify the key by offsetting (i.e., XOR-ing with any value) the masked key expression. Nonetheless, the key extracted from such "mangled" expression remains unknown to the attacker.

6.3 *Insider* Attacks

6.3.1 Key Modification and Re-use

The new danger introduced in this 3PKDP as a result of using the same key in messages 2 and 4 are the so-called *insider* attacks by either A or B. Both A and B, being privy to K_{ab}, can discover each other's $AUTH$ expressions and try to use this new knowledge in some malicious fashion.

Knowing K_{ab}, A (or B) can now alter B's (or A's) key distribution token to any desired value. Whether or not this is a *real* threat depends on the requirements specific to the local environment. The present protocol certainly fulfills the requirements of non-disclosure, non-modification, non-reuse and independence as defined in section 5.1, as long as the adversary is an outsider. In its current state, the protocol is vulnerable to modification or reuse of K_{ab} by an *insider* (A or B). In other words, neither of the two parties can be sure that the K_{ab} was actually issued by the AS.

This exposure cannot be addressed without changing the original 2PKDP. (See section 6.4 below.)

6.3.2 Other Attacks

Insider attacks can take on a broader scope. In general, any legitimate party X that is registered at the AS, can use 3PKDP to obtain a set of messages of the form:

$$AUTH_{K_{as}}(N_{as}, N_{sa}, X) \oplus K_{ax},\ N_{sa}$$
$$AUTH_{K_{xs}}(N_{xs}, N_{sx}, A) \oplus K_{ax},\ N_{sx}$$

for any selected party A and any selected values of N_{xs} and N_{as}. Because X can extract K_{ax} from the second message, he can also obtain the corresponding $AUTH$ expression: $AUTH_{K_{as}}(N_{as}, N_{sa}, X)$.

Since the AUTH expression is computed using K_{as}, the only way X could try to misuse this information is in an attempt to modify or find a key distributed to

[3] We note that the problem of malicious *insiders* does not exist in two-party key distribution.

[4] A more *sophisticated* protocol would optimize the number of protocol flows and minimize the size of each flow. At this point, however, we are not yet concerned with optimization.

[5] This version of the protocol only intends to show the properties of the key distribution, and doesn't deal with synchronizing A and B (how does B know that A wants to communicate?). In a real protocol, this issue could be dealt with by having A send all its communication with AS over B. This would imply that A would start by contacting B instead of AS.

A in a 3PKDP execution between A and some other party, say B. For such an attempt to succeed, X needs to compute $AUTH_{K_{as}}(N_{as}, N_{sa}, B)$ for either: i) a random value of N_{as} or ii) a random value of N_{sa}. Since X only knows expressions of the form $AUTH_{K_{as}}(\ldots, \ldots, X)$, finding an expression of the form $AUTH_{K_{as}}(\ldots, \ldots, B)$ is equivalent to breaking 2PAP.

6.4 Maintaining Key Integrity

In order to protect against insider attacks and maintain the integrity of the new key, we need to modify 2PKDP and, consequently, 3PKDP to explicitly include the integrity check of the key being distributed. The protocol in Figure 4 is essentially the same as the basic protocol in Figure 2 except that N_{qp} is no longer a simple nonce, but the encryption[6] of the other party's nonce under the new session key, i.e., $N_{qp} = E_{K_{new}}(N_{pq})$.

For the sake of simplicity, we focus on the two-party version in the rest of this section, however, the discussion is equally applicable to a three-party variation.

$P \Longrightarrow$ Q	P, N_{pq}
$Q \Longrightarrow$ P	$AUTH_{K_{pq}}(N_{pq}, E_{K_{new}}(N_{pq}), Q) \oplus K_{new}$, $E_{K_{new}}(N_{pq})$

Figure 4: Two-Party Key Distribution with Self-Checking Nonces

The implications of this change are not immediately obvious. An integrity check benefits the receiver of the message but it can also be of value to an attacker. For example, an outsider attacker now possesses cleartext-ciphertext pairs of the form $[N_{as}, E_{K_{ab}}(N_{as})]$. This new information can help in breaking the session key via a known plaintext attack. On the other hand, the session key is no longer subject to insider modification. Any change in the expression containing the key must be accompanied by the change in the corresponding key integrity check. For example, if an (in- or outsider) intruder wants to distribute K'_{ab} to A instead of K_{ab}, it must be able to change the key distribution message for A from:

$$AUTH_{K_{as}}(N_{as}, E_{K_{ab}}(N_{as}), B) \oplus K_{ab},\ E_{K_{ab}}(N_{as})$$

to:

$$AUTH_{K_{as}}(N_{as}, E_{K'_{ab}}(N_{as}), B) \oplus K'_{ab},\ E_{K'_{ab}}(N_{as})$$

Equivalently, the integrity of the key is dependent on the intruder's ability to generate a pair of values:

$$X = E_{K'_{ab}}(N_{as})$$

and

$$Y = AUTH_{K_{as}}(N_{as}, E_{K'_{ab}}(N_{as}), B) \oplus K'_{ab}$$

[6] Or the result of applying any strong trapdoor one-way function.

where K_{as}, N_{as} and B are *fixed* and K'_{ab} is *free*.[7]

If we assume that the encryption function is one-to-one, i.e., the encryption of N_{as} under a given key (K_{ab}) is distinct, then we can infer that Y is a function X. This is because X uniquely determines the value of K_{ab} which, in turn, uniquely determines the value of $AUTH_{K_{as}}(N_{as}, E_{K'_{ab}}(N_{as}), B)$. Making a further assumption that $AUTH$ is based on a secure MAC allows us to conclude that, since both N_{as} and B are *fixed*, the value of $E_{K'_{ab}}(N_{as})$ is uniquely determined from

$$AUTH_{K_{as}}(N_{as}, E_{K'_{ab}}(N_{as}), B).$$

Hence, for every possible choice of Y there exists only one value of X.

Suppose that the intruder is able to compute a pair of values $[X, Y]$ satisfying the above conditions. As a result of this computation, the intruder must be able to verify the relationship between X and Y. This verification **cannot** be achieved without the knowledge of K'_{ab}. But, since

$$Y \oplus K'_{ab} = AUTH_{K_{as}}(N_{as}, E_{K'_{ab}}(N_{as}), B)$$

the intruder simultaneously comes into possession of

$$AUTH_{K_{as}}(N_{as}, E_{K'_{ab}}(N_{as}), B)$$

However, knowledge of this $AUTH$ expression *contradicts* our assumptions about a secure 2PAP. The intruder can obtain an $AUTH$ expression in one of two ways:

1. by obtaining a "fresh" $AUTH$ expression that has never been computed by S.
   or
2. by obtaining "stale" $AUTH$ expressions previously used by S in a past run of the protocol.

Case 1: is in direct contradiction to the properties of a secure 2PAP as described in Sections 4 and 5.2. Case 2: any $AUTH$ expression from a previous protocol run is bound to the nonce N_{as} used in that execution. Since A can be expected to generate good nonces in every new protocol run, it is impossible to reuse stale $AUTH$ expressions. Therefore, the attack is impossible.

7 Conclusions

This paper discussed the issues of secure key distribution in a network environment. Results of previous research in secure two-party authentication were used to construct a range of simple and secure protocols for both two- and three-party key distribution. The proposed protocols are compact and require minimal additional computation on top of existing authentication

[7] The term *fixed* means that the variables involved are set for the duration of the protocol run; *free* means that the variable is allowed to take on any value.

protocols. This makes them suitable for adaptation at any layer of the network protocol hierarchy; in particular, at the data link and network layer where both bandwidth and computational resources are often limited.

8 Ongoing Work

The integrity of the key being distributed has not been a major concern in the design of the basic protocols presented in this paper. However, it is recognized that in many cases, key integrity is a required property. The solution proposed in section 6.4, using self-checking nonces, preserves key integrity but opens the door for known-plaintext attacks on the key being distributed. This observation led to the development of a somewhat improved protocol variant, which protects the integrity of the key without exposing it to known-plaintext attacks. This is achieved without increasing either the message sizes or the number of cryptographic operations. Efforts are currently under way to demonstrate the security of the improved protocol variant.

9 Acknowledgements

This work benefited from discussions with Phil Janson and Ralf Hauser and comments of the anonymous referees.

References

[1] S.M. Bellovin, M. Merritt, *Limitations of the Kerberos Authentication System*, ACM SIGCOMM Computer Communication Review, October 1990.

[2] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, *Systematic Design of Two-Party Authentication Protocols*, Proceedings of CRYPTO'91, August 1991.

[3] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, *A Modular Family of Secure Protocols for Authentication and Key Distribution* Draft, in submission to IEEE/ACM Transactions on Networking, August 1992.

[4] M. Burrows, M. Abadi, and R. Needham, *A Logic of Authentication*, **Proceedings of ACM Symposium on Operating System Principles**, December 1989.

[5] D. Denning and G. Sacco, *Timestamps in Key Distribution Systems*, **Communications of the ACM**, August 1981.

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

[7] Li Gong, *Using One-Way Functions for Authentication.* ACM Computer Communications Review. October 1989.

[8] L. Gong, R. Needham and R. Yahalom, *Reasoning About Belief in Cryptographic Protocols*, Proceedings of IEEE Symposium on Security and Privacy, May 1990.

[9] R. Molva, G. Tsudik, E. Van Herreweghen, S. Zatti, KryptoKnight *Authentication and Key Distribution Service*, Proceedings of ESORICS 92, October 1992; Toulouse, France.

[10] National Bureau of Standards, *Federal Information Processing Standards*, National Bureau of Standards, Publication 46, 1977.

[11] R. Needham and M. Schroeder, *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM, December 1978.

[12] R. Needham and M. Schroeder, *Authentication Revisited*, ACM Operating Systems Review, Vol. 21, No. 7, January 1987.

[13] R. Rivest, *The MD5 Message Digest Algorithm*, Internet DRAFT, July 1991.

[14] R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM, February 1978.

[15] J. Steiner, *The Kerberos Network Authentication Service Overview*, MIT Project Athena RFC, Draft 1, April 1989.

[16] J. Steiner, C. Neuman, J. Schiller, *Kerberos: An Authentication Service for Open Network Systems*, Proceedings of USENIX Winter Conference, February 1988.

[17] G. Tsudik, *Message Authentication with One-Way Hash Functions*, Proceedings of IEEE INFOCOM 1992. May 1992.