Using Signcryption to Build Compact and Efficient Protocols for Unforgeable Session Key Establishment *

Yuliang Zheng
School of Computing and Information Technology
Monash University
Melbourne, VIC 3199, AUSTRALIA
Email: yuliang.zheng@monash.edu.au

Hideki Imai
Institute of Industrial Science
The University of Tokyo
7-22-1 Roppongi, Minato-ku, Tokyo 106-8558, JAPAN
Email: imai@imailab.iis.u-tokyo.ac.jp

March 29, 1999

Abstract

Authenticated session key establishment is a central issue in network security. With the rapid deployment of the information superhighway which uses ATM or Asynchronous Transfer Mode for data transmission, compactness and efficiency have emerged as critical factors that have to be taken into account in the design of a key establishment protocol. This paper addresses a question on whether we can design a compact, efficient and authenticated key establishment protocol that has the following two properties: (1) each message exchanged between two participants can be transferred in a short data packet, and (2) messages that carry key materials are unforgeable and non-repudiatable without the involvement of a trusted key distribution center. We discuss why the answer to this question is negative if one follows the currently standard approach to key establishment, namely employing secret/public key encryption and, possibly, digital signature. We then present a number of protocols that represent a positive answer to the question, which is followed by extensions of the protocols to secure multicast in which a multiple number of recipients are involved. Our protocols are all based on a recently introduced cryptographic primitive called "signcryption" that fulfills both the functions of digital signature and public key encryption with a cost far smaller than that required by "digital signature followed by encryption".

Key Words

ATM Networks, Cryptography, Key Establishment, Multicast, Network Security, Signcryption

*An extended summary of this paper has appeared as "Compact and unforgeable key establishment over an ATM network", in the Proceedings of IEEE INFOCOM'98, pp.411-418, 1998.

1 Introduction

A key establishment protocol is a sequence of specified steps between two or more participants whereby the participants can agree on a shared secret value. The shared secret value is called a *session key*, due to the fact that it is usually employed for a single communication session and hence lives for only a relatively short period of time. A major motivation behind session key establishment is to cryptographically eliminate correlations across different communication sessions, which would minimize security exposure when a particular session key is compromised. Cryptographic independence of communication sessions would also significantly reduce the risk of replay attacks by an active attacker who has recorded past communication sessions and tries to compromise a current communication session by inserting into it, or replacing (part of) it with, (part of) past sessions. The attack may have or have not compromised the contents of past communication sessions.

A key establishment protocol falls into one of two types. Protocols in the first type rely on shared static keys and use secret key (or symmetric) cryptosystems to ensure the confidentiality of message contents. Although such protocols are generally very efficient, potential problems with them include those associated with the generation and management of static keys. In contrast, protocols in the second type employ public key (or asymmetric) cryptographic techniques. These protocols do not have the problems with static keys, but are not as efficient as those based on secret key cryptosystems.

Motivated by a specific problem arising from a research project on Video on Demand over an Asynchronous Transfer Mode (ATM) network, the aim of this work is to design key establishment methods that (1) are efficient, i.e., of a low computational cost, (2) are compact so that a message can be fitted into a small data packet such as a single ATM cell, and (3) offer message unforgeability and non-repudiation, without the involvement of a trusted key distribution center.

After a thorough analysis of well-known key establishment protocols based on public key cryptography in the literature, we show that none of these protocols fulfills all the three requirements. Furthermore, as the major contribution of this paper we show how signcryption, a recently discovered new primitive in public key cryptography, can be used to construct a number of key establishment protocols all of which fulfill the three requirements. Another contribution of this work is to extend the protocols to multicast conference key establishment in which a participant wishes to agree on a common secret key with a multiple number of recipients. We envisage that these protocols will find applications not only in high speed network layer security solutions, but also in less demanding application layer solutions.

The remainder of this paper is organized as follows: Section 2 examines a number of important issues in key establishment. This is followed by a statement of the motivation and goals of the research in Section 3. The signcryption primitive and an concrete implementation of the primitive are described in Section 4, and basic data formats for transporting key materials using signcryption are introduced in Section 5. Section 6 highlights the contributions of this paper by presenting a set of key establishment protocols built on the data formats in the previous section. This is followed by an analysis of the proposed protocols and a comparison between them and existing solutions in Section 7. Section 8 extends our protocols to multicast conference key establishment. The paper is closed with some concluding remarks in Section 9.

2 Dimensions in Key Establishment

There has been an extremely large body of research in the area of key establishment since the publication of the landmark paper by Diffie and Hellman [10], which has resulted in a situation where one may find numerous protocols in the literature, each of which may have different properties. A primary reason behind the emergence of such a large number of key establishment protocols can perhaps be attributed to the many different dimensions of key establishment. The rest of this section is intended as a brief summary of the many facets in key establishment, with a view to properly position this work in the research area. The reader who wishes to find more detailed information on various key establishment methods proposed so far may consult Chapter 12 of [21] and references listed in the handbook.

To simplify our discussions in this section, we will primarily focus on key establishment between two participants. It is, however, straightforward to extend the discussions to the case of key establishment among three or more. Following a tradition in cryptography, one participant will be called Alice and other Bob. The term of "a participant" is intended to be a general one to represent a user, a computing process, a communication device, a general information processing machine and so on.

2.1 Security

A session key established by an execution of a protocol should be known only to the two participants involved, and also to a KDC or key distribution center if the protocol involves the KDC. Security of the session key should not be compromised under all the possible attacks that might be encountered in a particular environment where the protocol will be employed. Typical attacks include (1) inferring a session key via (passive) eavesdropping, (2) replaying past messages, (3) interleaving messages from one protocol execution with another, (4) deducing a session key with a known past session key.

To defend against replay attack, a session key has to be fresh and new. Freshness is usually achieved through the use of a time-varying quantity, such as a time-stamp or a nonce (random number) or a combination of both. The main difference between the use of a time-stamp and a nonce is that the former assumes the existence of a synchronized clock between two participants (and/or between participants and a KDC in a KDC-based protocol). While the requirement of a synchronized clock in a local area network should be readily fulfilled, it may pose a problem in a wide area network.

Interleaving attacks were extensively studied in [4]. In an attempt to formalize the notion of resisting against replay and interleaving attacks, a concept of "matching of protocol histories" was also introduced in [4]. The concept was extended to the notion of "matching protocol runs" in [11]. The latter was further refined to a general notion in [2] where the authors presented the first formal security proofs for two authenticated key establishment protocols. In the complexity-theoretic model proposed in [2], an attacker has full control over all the transactions among participants. In particular, the attack is allowed to engage himself in as many executions of a key establishment protocol as he wishes at any particular point of time.

In addition to the above attacks, some applications may require that a key establishment protocol have the capacity of minimizing the impact of compromise of a long term secret key. This requirement is called "(perfect) forward secrecy". A key establishment protocol is said to offer forward secrecy with respect to a particular participant if compromise of the participant's long term secret key does not result in the exposure of past session keys.

Clearly, a secret key based protocol cannot offer forward secrecy with respect to a participant or a KDC whose long term secret key is involved in encrypting key materials. The same can be said with a public key based protocol, with respect to a participant whose long term private key is used to extract key materials. If a participant's private key is only involved in the creation of a digital signature on messages, compromise of the private key may not directly expose a session key, rather it may result in impersonation of the participant by an attacker. The Internet Oakley key determination protocol [25, 14] has been designed to achieve forward secrecy. The reader is referred to Section 7.2 for a detailed discussion on more economical techniques for reducing the chance that a long term secret key might be compromised.

Another relevant issue is on formal proofs or arguments of security. Protocols that have been designed to resist against known attacks and have survived these attacks are said to admit heuristic security. Since such protocols are not proved to be secure against *all* attacks that may arise in practice, many of them have been found to be flawed [6]. Therefore ideally one would like to have protocols that admit *provable security* against all attacks. A complexity-theoretic approach towards provable security was initiated in [2] where the provable security of two key establishment protocols based on secret key cryptosystems was also presented.

2.2 Authentication

Entity authentication is a process by which a participant is convinced of the identity of another participant. Entity authentication can be unilateral (one-way) or mutual (two-way). In a mutual authentication protocol, both participants wish to be convinced that the other participant is indeed who he/she claims to be.

A concept that is closely related to and often confused with entity authentication is *identification*. While the aim of identification is similar to entity authentication, namely for one participant, say Alice, to convince another participant, say Bob, of her identity, identification satisfies a more stringent requirement: no participant other than Alice can prove that he or she is Alice, even to him or herself. The difference between entity authentication and identification is made clear by examining a protocol based on a shared static key between Alice and Bob. Alice and Bob can mutually authenticate each other using the static key in three moves [16]. However, such a protocol is not an identification protocol, since whatever produced by Alice using the shared key can also be created by Bob, and vice versa.

An implicit requirement of key establishment in practical applications is that it offers at least unilateral entity authentication or identification. A protocol that offers both key establishment and (unilateral/mutual) entity authentication or identification is called a (unilaterally/mutually) authenticated key establishment.

In secret key cryptosystem based protocols, entity authentication is almost exclusively achieved by way of challenge-and-response. A key establishment protocol employing public key cryptosystems can offer identification by the use of digital signature.

Technically, some entity authentication or identification protocols may be modified to carry session keys, and conversely, an authenticated key establishment can also be used for the purpose of entity authentication or identification.

Finally there is another concept related to authentication. That is key confirmation whose purpose is for a participant to acknowledge that he or she is indeed in possession of a session key. The last message of a key establishment protocol can be optionally composed

of an authentication tag. The tag may be generated from a newly agreed session key by $tag = MAC_{key}(known\_message)$, where MAC may be instantiated with either a message authentication code or a keyed hash function.

2.3 Unforgeability and Non-repudiation

In some applications, a participant may require that his or her messages cannot be forged by other participants. Symmetrically, the recipient of a message, especially of one that contains key materials, may require that the sender of the message cannot repudiate at a later stage the fact that he or she is the originator of the message. We envisage that in electronic commerce, non-repudiation and unforgeability of key materials and actual communication sessions that employ a key derived from the key materials is of particular importance.

Both unforgeability and non-repudiation can be achieved by using digital signature. With a secret key based protocol, however, neither unforgeability nor non-repudiation can be achieved, unless the protocol involves a KDC, possibly together with tamper-resistant devices.

Unforgeability and non-repudiation are closely related to identification. In particular, unforgeability or non-repudiation cannot be accomplished unless a key establishment protocol is also an identification protocol.

Let us turn our attention to high speed networks. In a (draft) document prepared by the ATM Forum [8], *accountability* for all ATM network service invocations and management activities, as well as for each individual entity's actions, has been identified as one of the four main security objectives, with the other three being confidentiality, data integrity and availability. This objective would not be achieved unless key establishment protocols employed fulfill unforgeability and non-repudiation.

2.4 Transport v.s. Exchange

We distinguish between two types of key establishment protocols: *key (material) exchange* protocols [1] and *key (material) transport* protocols. With a key exchange protocol, a shared session key is derived from joint key materials from both participants. Such a protocol requires both participants involved to exchange key materials. In contrast, with a key transport protocol, key materials from which a session key is derived are created by one participant and transferred to the other. A key exchange protocol may be preferred to a key transport protocol in certain applications where a session key is required to be "fair", in that it is dependent on both participants' key materials. However, one should distinguish between key material exchange and shared generation of random numbers as achieved in threshold cryptography [9]. In particular, with a key exchange protocol a participant who is in a position to see, prior to producing his key materials, those from the other participant may control the resultant session key by carefully choosing his key materials. In this sense, a key (material) exchange protocol is essentially the same as a key (material) transport protocol. In general, truly "fair" session key generation cannot be achieved without the involvement of computationally expensive bit/sequence commitment, and hence it is these authors' view that "fairness" should not be set as a goal of key establishment.

Some examples of key transport protocols are Kerberos [19] and X.509 strong authenticated protocols [15]. The most prominent representative of key exchange protocols is the Diffie-Hellman protocol [10].

[1] Key exchange protocols are also called key *agreement* protocols by some researchers.

2.5 Secret v.s. Public Key Cryptosystems

Prior to the execution of a key establishment protocol, two participants may or may not have shared static keys in their hands. In the case of having a shared static key, the most efficient way for them to establish a fresh session key is to use a key establishment protocol built on a secret key (or symmetric) cryptosystem. Both Kerberos [19] and KryptoKnight [5, 16] are based on secret key cryptosystems.

A shared static key may have been established in three different ways: (1) an explicit key established previously, (2) an implicit key defined in a key pre-distribution scheme [20], and (3) an implicit key defined in the Diffie-Hellman protocol [10].

On the other hand, if the two participants do not have a shared static key, they may have to use a public key cryptosystem which is not as efficient as a secret key cryptosystem, unless they can ask for help from a key distribution center with whom both participants have a separately shared static key. Examples of key establishment using public key cryptosystems are X.509 strong authenticated key transport protocols [15], Beller-Yacobi protocols [3] and the Internet Oakley key determination protocol [25, 14]. The latter is a modified version of a station-to-station protocol proposed in [11].

Note that for efficiency reason, a public key based protocol usually uses secret key cryptosystems in encrypting data. Other cryptographic primitives such as authentication codes and one-way or keyed hashing may also be used in combination with a secret or public key based protocol.

2.6 Trusted Third Party

In some applications, there may be a trusted third party called a key distribution center (KDC), with which each participant may have a shared static key. When the KDC can be involved in the process of session key establishment, there may be no need for participants to have shared static keys among themselves. In this sense, the availability of a KDC facilitates or simplifies key establishment. A disadvantage of the involvement of a KDC is that compromise of the KDC would result in the compromise of an entire system that employs it. In addition, the KDC may become a bottleneck of the system, due to a heavy load imposed on it.

Notable examples of key establishment relying on a KDC include Kerberos [19] and some versions of KryptoKnight [5, 16].

In some other applications, there may be no KDC available. Instead, there may exist only an "off-line" trusted third party called a certification authority (CA). The main function of a certification authority is to issue certificates on a participant's public key(s) by the use of digital signature. Although a certification authority does not involve itself in the process of key establishment, its existence is implicit in all public key based session key establishment protocols.

2.7 Efficiency

Each application may have its own set of requirements on the efficiency of a key establishment protocol. For example, secure mobile communications generally require a "light-weight" protocol, as a mobile device is usually computationally less powerful than a wired one. As a second example, a network layer security application has far more stringent requirements on the efficiency of key establishment than does an upper layer application.

Factors that contribute to the efficiency of a key establishment protocol include (1) the number of moves (or flows, passes) of messages between two participants, (2) the length of messages communicated between the participants (measured in bits), (3) the computational cost invested by both participants, (4) the size of secure storage, (5) the degree of pre-computation (which is especially important if the protocol is intended to be used with computationally weak devices), and so on. One of the challenges that face a protocol designer is to arrive at a key establishment protocol that would not only minimize the first four factors but also maximize the fifth factor, while maintaining the goals of the protocol.

Optimally efficient protocols that rely on secret key cryptosystems and/or a KDC have been proposed in [5, 13]. Among public key based protocols, Beller and Yacobi's proposals [3] minimize the computational requirements of a less powerful participant. These protocols are particularly suitable for applications where one of the two participants is computationally weak.

3 Goals and Motivation of This Research

The main goals of this research are to design authenticated key establishment protocols that (1) do not rely on a trust key distribution center or KDC, (2) have a low computational cost, (3) are compact so that the length of each message exchanged is as short as possible, and (4) offer unforgeability and non-repudiation.

As will be shown in the forthcoming sections, we will achieve the goals by proposing a set of specific protocols that satisfy all the four conditions. While these new key establishment protocols can aide to solve security problems in a wide range of areas, it is instructive to mention that this work has been partially inspired by a research project on Video on Demand over an insecure ATM network. With ATM, data are carried in cells, each of which is composed of a 5-byte header and a 48-byte payload field. Only the 48-byte or 384-bit payload can be used for transmitting data. the 5-byte header is reserved for carrying control information. Transmitting a data item of more 384 bits over an ATM network would require two or more ATM cells. While ATM networks are significantly faster than most networks widely used today, transmitting a data item across two or more cells would result in a delay that may not be tolerable in certain high speed applications, primarily due to the necessity of data packetization, buffering, and re-assembling. Therefore, ideally one would like to transmit encrypted key materials in a single ATM cell without the need of splitting data. The research project on Video on Demand involves a system that can be modeled by a server acting as a content provider, together with a large number of clients connected to the server via an insecure ATM network. When a client wishes to view a specific video clip, it would send a short request to the server, indicating which video clip it wishes to receive. As there are a large number of clients who are assumed to be mutually untrustworthy, one issue related to a request is unforgeability and non-repudiation. Another is confidentiality of a video clip transmitted over the insecure network. In addition, as many of the clients may make a request at the same point of time, to maintain the required Quality of Services, it is crucial that each request is processed as swiftly as possible both by the client and the server. The project team realize that it would be nice if a request could be (1) efficiently assembled by a client, (2) packed into a single ATM cell, carrying all the information, including a video clip identifier and a key to be used in encrypting the clip, in a secure and authenticated fashion, and (3) efficiently disassembled by the server.

In many key transport protocols that rely on secret key cryptosystems, such as those

proposed in [2, 5], messages communicated between Alice and Bob are all compact and can be easily fitted into single ATM cells. Some of these protocols do not offer unforgeability or non-repudiation, while the others do so only with the help of a KDC. In other words, these protocols are not suitable for an application where unforgeability and non-repudiation are to be satisfied without relying on a KDC.

Key establishment using public key cryptosystems does not rely on a KDC in achieving unforgeability and non-repudiation. With all currently known public key based key establishment protocols, however, a single payload field of 48 bytes, or of 384 bits, cannot be used to carry unforgeable key materials. To see why this is the case, we take the RSA cryptosystem as an example. In order to maintain a minimal level of security, it is widely believed that the size of an RSA composite should be of at least 512 bits. Thus merely encrypting key materials will result in an expanded outcome that has as many bits as in the RSA composite. (See [17] for a discussion on various data formats for key transport using RSA.) If, in addition, digital signature is involved to achieve unforgeability, the outcome will be even longer. A similar problem occurs with public key cryptographic techniques based on the ElGamal encryption scheme that relies on the discrete logarithm over finite fields.

The ElGamal encryption scheme built on an elliptic curve over a finite field, say $GF(2^{160})$, deserves special attention. With this scheme, a point on the elliptic curve can be compressed so that it occupies only $160+1 = 161$ bits. Thus a single ATM cell may be used to transmit un-authenticated key materials of up to about $384 - 161 = 223$ bits. However, a field of 223 bits is too small to carry a key and a time-varying quantity together with a signature. In other words, elliptic curve based public key cryptography does not provide a solution to the problem of compact and unforgeable key establishment.

In the following sections, we show how a recently proposed cryptographic primitive called signcryption can be used to achieve the seemingly impossible goal, namely, to transmit secure and unforgeable key materials in a single ATM cell. To close this section, let us stress once again that although the problem with carrying key materials in a single ATM cell may be of interest only to the specific research project on Video on Demand, due to the fact that in practice data is usually subject to encapsulation ("extension") at various layers while traveling down the ATM protocol stack, the spectrum of applications of the key establishment protocols to be introduced in the following sections is broad and by no means limited only to the project on Video on Demand.

4 Signcryption

A *signcryption* scheme is a cryptographic method that fulfills both the functions of secure encryption and digital signature, but *with a cost smaller than that required by signature-then-encryption.* In the following we show an example implementation of signcryption based on the infeasibility of computing discrete logarithm over a large finite field. The example signcryption scheme is called SCS1 and it uses a shortened version of the Digital Signature Standard [23]. The reader is directed to [28, 30] for other example implementations of signcryption.

In describing our method, we will use E and D to denote the encryption and decryption algorithms of a private key cipher such as DES [22] and SPEED [29]. Encrypting a message m with a key k, typically in the cipher block chaining (CBC) or output feedback (OFB) mode, is indicated by $E_k(m)$, while decrypting a ciphertext c with k is denoted

by $D_k(c)$. Note that in the OFB mode, the encryption algorithm effectively serves as a cryptographically secure pseudo-random sequence generator. In addition we use $KH_k(m)$ to denote hashing a message m with a keyed hash function/algorithm KH under a key k. An important property of a keyed hash function is that, just like a one-way hash function, it is computationally infeasible to find a pair of messages that are hashed to the same value (or collide with each other). This implies a weaker property that is sufficient for signcryption: given a message m_1, it is computationally intractable to find another message m_2 that collides with m_1. In [1] two methods for constructing a cryptographically strong keyed hash algorithm from a one-way hash function/algorithm have been demonstrated. For most practical applications, it suffices to define $KH_k(m) = hash(k, m)$, where $hash$ is a one-way hash function. SHS [24] and HAVAL [31] are two examples of one-way hash functions.

Let p be a large prime, q a large prime factor of $p-1$, and g an integer with order q modulo p chosen randomly from $[1, \ldots, p-1]$. Assume that Alice has chosen a private key x_a from $[1, \ldots, q-1]$, and made public her matching public key $y_a = g^{x_a} \bmod p$. Similarly, Bob's private key is x_b and his matching public key is $y_b = g^{x_b} \bmod p$. Relevant public and private parameters are summarized in Table 1.

Parameters public to all:
p — a large prime
q — a large prime factor of $p-1$
g — an integer with order q modulo p chosen randomly from $[1, \ldots, p-1]$
$hash$ — a one-way hash function whose output has, say, at least 128 bits
KH — a keyed one-way hash function
(E, D) — the encryption and decryption algorithms of a private key cipher
Alice's keys:
x_a — Alice's private key, chosen uniformly at random from $[1, \ldots, q-1]$
y_a — Alice's public key ($y_a = g^{x_a} \bmod p$)
Bob's keys:
x_b — Bob's private key, chosen uniformly at random from $[1, \ldots, q-1]$
y_b — Bob's public key ($y_b = g^{x_b} \bmod p$)

Table 1: Parameters for Signcryption

Table 2 details the signcryption scheme SCS1. As shown in the table, the signcryption and unsigncryption algorithms for SCS1 are remarkably simple. The signcrypted version of a message m is composed of three parts c, r and s from which the recipient can recover the original message. Note that in the table, $\in_R$ indicates an operation that chooses an element uniformly at random from among a set of elements.

With the signcryption scheme SCS1, *bind_info* may contain, among other data, the public keys or public key certificates of both Alice the sender and Bob the recipient. It is included in the creation of r for the purpose of tightly linking a message m to participants involved. The output of the one-way hash function $hash$ used in defining $(k_1, k_2) = hash(y_b^x \bmod p)$ should be sufficiently long, say of at least 128 bits, which guarantees that both k_1 and k_2 have at least 64 bits. Also note that in practice, (k_1, k_2) can be defined in a more liberal way, such as $(k_1, k_2) = y_b^x \bmod p$ and $(k_1, k_2) = fd(y_b^x \bmod p)$, where fd denotes a folding operation.

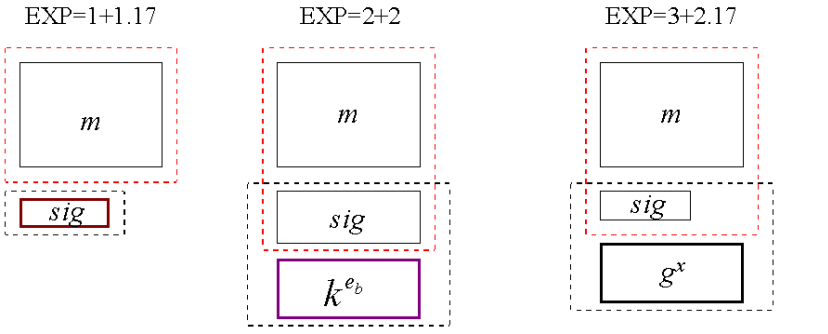
The unsigncryption algorithm works by taking advantages of the property that $g^x \bmod p$ can be recovered from r, s, g, p and y_a by Bob. It should be noted that signcryption schemes

Signcryption of m by Alice the Sender		**Unsigncryption of (c, r, s) by Bob the Recipient**
$x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ $c = E_{k_1}(m)$ $r = KH_{k_2}(m, bind\_info)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $m = D_{k_1}(c)$ Accept m only if $KH_{k_2}(m, bind\_info) = r$

Table 2: An Example Implementation of Signcryption (SCS1)

can also be derived from ElGamal-based signature schemes built on other versions of the discrete logarithm problem such as that on elliptic curves [18].

The format of the signcrypted text of a message m, together with the formats of the traditional "signature-then-encryption", is depicted in Figure 1. The reader will notice that a notation $EXP = N_1 + N_2$ is used in the figure. N_1 indicates the number of modular exponentiations carried out by a sender, and N_2 indicates the number by a recipient. Table 3 summarizes the advantage of SCS1, in terms of savings in computational cost and communication overhead, over discrete logarithm based signature-then-encryption, while Table 4 summarizes that over RSA based signature-then-encryption.



(a) Signcryption based on DL (b) Signature-then-Encryption based on RSA (c) Signature-then-Encryption based on DL

Figure 1: Output Formats of Signcryption and Signature-then-Encryption

5 Basic Ideas in Using Signcryption for Key Transport

Having introduced an example implementation of signcryption in the previous section, now we show how such an implementation allows transportation of key materials in an efficient and compact way. Messages exchanged are so compact that they can all be carried by *a single block whose size is smaller than* $|p|$. We present two possible data formats for Alice to transport key materials to Bob, one carrying directly while the other indirectly key materials.

security parameters			saving	saving in
$\|p\|$	$\|q\|$	$\|KH_.(\cdot)\|$	average comp. cost	comm. overhead
512	144	72	58%	70.3%
1024	160	80	58%	81.0%
2048	192	96	58%	87.7%
4096	256	128	58%	91.0%
8192	320	160	58%	94.0%
10240	320	160	58%	96.0%

saving in average comp. cost $= \frac{(5.17-2.17) \text{ modular exponentiations}}{5.17 \text{ modular exponentiations}} = 58\%$

saving in comm. cost $= \frac{|hash(\cdot)|+|q|+|p|-(|KH_.(\cdot)|+|q|)}{|hash(\cdot)|+|q|+|p|}$

Table 3: Saving of Signcryption over Signature-Then-Encryption Using Schnorr Signature and ElGamal Encryption

security parameters			advantage in	advantage in
$\|p\|(=\|n_a\|=\|n_b\|)$	$\|q\|$	$\|KH_.(\cdot)\|$	average comp. cost	comm. overhead
512	144	72	0%	78.9%
1024	160	80	32.3%	88.3%
2048	192	96	59.4%	93.0%
4096	256	128	72.9%	95.0%
8192	320	160	83.1%	97.0%
10240	320	160	86.5%	98.0%

advantage in average comp. cost $= \frac{0.375(|n_a|+|n_b|)-3.25|q|}{0.375(|n_a|+|n_b|)}$

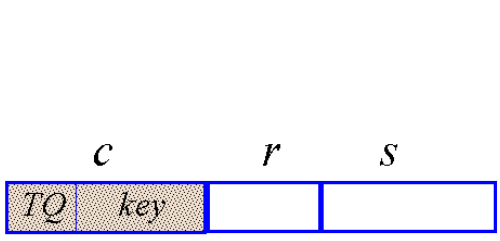
advantage in comm. cost $= \frac{|n_a|+|n_b|-(|KH_.(\cdot)|+|q|)}{|n_a|+|n_b|}$

Table 4: Advantage of Signcryption over RSA based Signature-Then-Encryption with *Small Public Exponents*

5.1 Direct Transport of Key Materials

Figure 2 illustrates a method for directly transferring key materials. It shows a possible combination of parameters: $|p| \geqq 512$, $|q| = 160$, and $|KH.(\cdot)| = 80$. The actual data from Alice to Bob consist of c, r and s, where $c = E_{k_1}(key, TQ)$, $r = KH_{k_2}(key, TQ, other)$ and $s = x/(r + x_a) \bmod q$, where the key part contained in (key, TQ) may be used directly as a random session key, TQ may contain a time-varying quantity such as a nonce or a time-stamp or both, and $other$ may be composed of the participants' identifiers, public key certificates and other supplementary information. It is preferable for E to act as a length-preserving encryption function so that (key, TQ) and $c = E_{k_1}(key, TQ)$ are of the same length.

Note that if key has 64 bits in length, and that TQ requires 32 bits, then $c = E_{k_1}(key, TQ)$ is of 96 bits, and (c, r, s) can be fitted even in a short packet that has only $96+80+160 = 336$ effective bits for data transport. Furthermore, if the quantity TQ is already known to Bob the recipient, then it may be dropped from $c = E_{k_1}(key, TQ)$ to save more bit locations for transferring key materials.



$|p| \geq 512,\ |q| \geq 160,\ |KH.(\cdot)| \geq 80$

$(k_1, k_2) = hash(y_b^{\ x} \bmod p)$
with $x \in_R [1, \ldots, q-1]$
$|k_1| \geq 64,\ |k_2| \geq 64$

$c = E_{k_1}(key, TQ)$
$r = KH_{k_2}(key, TQ, other)$
$s = \dfrac{x}{r + x_a} \bmod q$

Figure 2: Direct Transport of Key Materials

5.2 Indirect Transport of Key Materials

In certain applications, part of a small packet may be used for other purposes, which would leave no room to directly accommodate both a random session key and a time-varying quantity. With such a short packet, we can transport (part of) key materials indirectly. In particular, we may define (c, r, s) as $c = E_{k_1}(TQ)$, $r = KH_{k_2}(TQ, other)$, and $s = x/(r + x_a) \bmod q$. (see Figure 3). The actual session key may be derived from (k_1, k_2) and other materials, through, for instance, the application of a keyed hash function.

Now assume that TQ has 32 bits. Then we can accommodate (c, r, s) using only $32 + 80 + 160 = 272$ bits. In the case where TQ is already known to Bob, the creation and transmission of the c part can be skipped.

Finally we note that in both Figures 2 and 3, a long TQ, say of 64 bits, may need not be encrypted. However, encryption is mandatory for a short TQ, say of $\leqq 40$ bits, in order to reduce the risk of replay attacks.

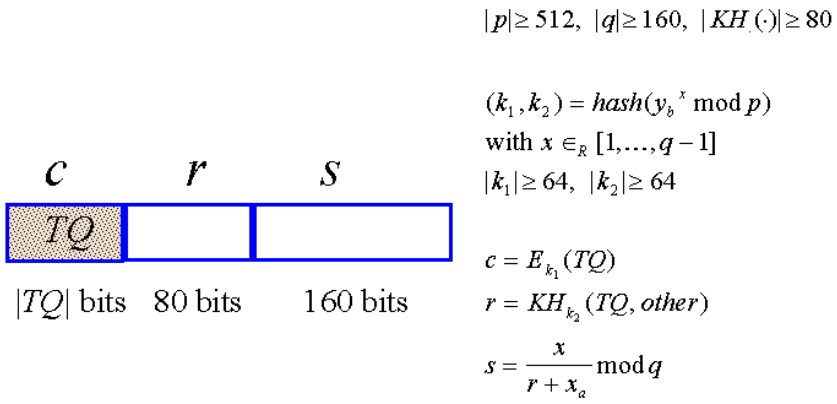


Figure 3: Indirect Transport of Key Materials

6 Signcryption Based Key Establishment

Now we are ready to describe in full details how to establish fresh random session keys between two participants Alice and Bob, in such a way that all messages exchanged between the two participants are short and computational costs involved are minimized.

6.1 Assumptions

In the following discussions, we assume that system parameters that are common to all participants, and the public and private keys of both Alice and Bob have all been properly set up according to Table 1. In addition, there is a trusted certification authority (CA) that has already issued a public key certificate to each participant. A participant's public key certificate may comply with X.509 certificate format that contains such information as certificate serial number, validity period, the ID of the participant, the public key of the participant, the ID of the CA, the public key of the CA, etc. It would be pointed out that the digital signature scheme used by the CA in creating public key certificates does not have to be one based on ElGamal signature scheme.

Furthermore, we assume that prior to an execution of a key establishment protocol, both participants have already obtained the other participant's public key and its associated certificate issued by the CA, and have checked and are satisfied with the validity of the certificates. The participants may have done so either because they both keep a list of frequently used certificates, or they have obtained and verified the certificates for previous communication sessions.

In describing a key establishment protocol, $key \in_R \{0,1\}^{\ell_k}$ indicates that key is an ℓ_k-bit number chosen uniformly at random. Similarly $NC_b \in_R \{0,1\}^{\ell_n}$ is a nonce chosen by Bob. And TS is a current time-stamp. Typically $\ell_k \geqq 64$, $\ell_n \geqq 40$, and the number of bits in TS may be decided by the accuracy of clock synchronization, as well as by the life span of a message containing the time-stamp. Finally a 64-bit authentication tag would be long enough for the purpose of key confirmation in most practical applications.

We consider key establishment both through key material transport and exchange.

6.2 Key Transport Protocols

A key transport protocol may use either a nonce or a time-stamp in guaranteeing freshness. The protocol may also transport key materials either directly or indirectly. So there are in total four possible combinations. Table 5 describes two direct key transport protocols, while Table 5 the corresponding two indirect key transport protocols.

The *etc* part may contain data known to both Alice and Bob. Such data may include the participants' names, public keys, public key certificates, protocol serial number, and so on. It may also contain system control information. Note that one of the purposes of sending *tag* is for key confirmation, namely for a participant (Bob) to show the other (Alice) that he does know the new session key. For a less critical application, a time-stamp TS may be transmitted to Bob in clear to further improve the computational efficiency of the protocols. In addition, if both time-stamps and nonces are available in an application, TS may be substituted with a combination of a time-stamp and a nonce.

As can be seen in the tables, protocols that rely on a nonce require one more message move than protocols that rely on a time-stamp.

Direct Key Transport Using a Nonce (Protocol DKTUN)		
Alice		**Bob**
	$\Leftarrow \quad NC_b \quad \Leftarrow$	$NC_b \in_R \{0,1\}^{\ell_n}$
$key \in_R \{0,1\}^{\ell_k}$ $x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ $c = E_{k_1}(key)$ $r = KH_{k_2}(key, NC_b, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $key = D_{k_1}(c)$ Accept key only if $KH_{k_2}(key, NC_b, etc) = r$
verify *tag*	$\Leftarrow \quad tag \quad \Leftarrow$ (optional)	$tag = MAC_{key}(NC_b)$

Direct Key Transport Using a Time-Stamp (Protocol DKTUTS)		
Alice		**Bob**
$key \in_R \{0,1\}^{\ell_k}$ $x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ Get a current time-stamp TS $c = E_{k_1}(key, TS)$ $r = KH_{k_2}(key, TS, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $(key, TS) = D_{k_1}(c)$ Accept key only if TS is fresh and $KH_{k_2}(key, TS) = r$
verify *tag*	$\Leftarrow \quad tag \quad \Leftarrow$ (optional)	$tag = MAC_{key}(TS)$

Table 5: Direct Key Material Transport with Signcryption

Indirect Key Transport Using a Nonce (Protocol IKTUN)		
Alice		**Bob**
	$\Leftarrow \quad NC_b \quad \Leftarrow$	$NC_b \in_R \{0,1\}^{\ell_n}$
$x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ $key = k_1$ $r = KH_{k_2}(key, NC_b, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $key = k_1$ Accept key only if $KH_{k_2}(key, NC_b, etc) = r$
verify tag	$\Leftarrow \quad tag \quad \Leftarrow$ (optional)	$tag = MAC_{key}(TS)$

Indirect Key Transport Using a Time-Stamp (Protocol IKTUTS)		
Alice		**Bob**
$x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ Get a current time-stamp TS $c = E_{k_1}(TS)$ $r = KH_{k_2}(TS, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $TS = D_{k_1}(c)$ Accept (k_1, k_2) only if TS is fresh and $KH_{k_2}(TS, etc) = r$
$key = KH_{k_1,k_2}(TS)$ verify tag	$\Leftarrow \quad tag \quad \Leftarrow$ (optional)	$key = KH_{k_1,k_2}(TS)$ $tag = MAC_{key}(TS, 1)$

Table 6: Indirect Key Material Transport with Signcryption

6.3 Key Exchange and Mutual Identification

In the key transport protocols described above, messages from Bob are not involved the creation of a session key. If one wishes that the session key is generated jointly by Alice and Bob, there are a few different ways that can be used to accomplish this. Here are some examples: (1) $key^* = KH_{key}(NC_b)$, (2) $key^* = KH_{key}(ID_b)$, and (3) $key^* = KH_{key}(NC_b, ID_b)$, where NC_b is a nonce generated by Bob, ID_b is Bob's identifier, and key^* denotes a session key that is jointly determined by information from both Alice and Bob.

Two common properties shared by the four protocols are: (1) Alice identifies herself to Bob (her message to Bob is fresh and unforgeable even by Bob), (2) Bob authenticates himself to Alice if the last response message *tag* is sent (*tag* is fresh and unforgeable by any third party). The protocols can be modified to achieve *mutual identification*: Alice sends to Bob fresh and unforgeable key materials and vice versa. We take as examples the two protocols for direct key transport. Modifications to the protocols are shown in Table 7. The modified protocols are direct key (material) *exchange* protocols that achieve mutual identification. A resultant $key \oplus key^*$ can be used as a fresh session key jointly generated by both participants.

The other two protocols for for indirect key transport can be modified in a similar way.

6.4 Two-Way Communications

For two-way communications, Alice and Bob may need to agree upon a pair of random session keys key_1 and key_2. A simple technique is to employ a pseudo-random number generator or a good hashing function to "extend" key into (key_1, key_2).

7 Analysis and Comparison

As our key establishment protocols described in Tables 5 and 6 are essentially message transport schemes using signcryption, security of key materials are guaranteed by the security of the signcryption scheme against chosen message attacks [28, 30]. After the successful establishment of a session key, Alice convinces Bob of her identify (the message from Alice is fresh and unforgeable even by Bob). In contrast, Bob can authenticate himself to Alice by sending a response message *tag* which is fresh and unforgeable by a third party (but can be generated by Alice). The four protocols can be modified using a method shown in Table 7 in order to achieve mutual identification, at the expense of more computation and message exchanges.

Freshness of a session key is assured through the use of a nonce or a time-stamp. When *tag* is sent, both Alice and Bob are assured that the other participant does know the fresh random session key. The protocols do not rely on a KDC. In addition, key materials transported from Alice to Bob are unforgeable, even by Bob the recipient. The materials are also non-repudiatable by Alice. In an event when Alice denies the fact that she was the person who created certain key materials, Bob can ask for help from a third party called a judge. Bob and the judge may follow a zero-knowledge protocol in settling the dispute [28, 30]. Similar discussions on non-repudiation are applicable to Bob for a modified protocol with mutual identification.

Direct Key Exchange Using a Nonce (Protocol DKEUN)		
Alice		**Bob**
	$\Leftarrow \quad NC_b \quad \Leftarrow$	$NC_b \in_R \{0,1\}^{\ell_n}$
$key \in_R \{0,1\}^{\ell_k}$ $x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ $c = E_{k_1}(key)$ $r = KH_{k_2}(key, NC_b, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	(k_1, k_2) $\quad = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $key = D_{k_1}(c)$ Accept key only if $\quad KH_{k_2}(key, NC_b, etc) = r$
(k_1^*, k_2^*) $\quad = hash((y_b \cdot g^{r^*})^{s^* \cdot x_a} \bmod p)$ $key^* = D_{k_1^*}(c^*)$ Accept key^* only if $\quad KH_{k_2^*}(key^*, key, etc) = r^*$	$\Leftarrow \quad c^*, r^*, s^* \quad \Leftarrow$	$key^* \in_R \{0,1\}^{\ell_k}$ $x^* \in_R [1, \ldots, q-1]$ $(k_1^*, k_2^*) = hash(y_a^{x^*} \bmod p)$ $c^* = E_{k_1^*}(key^*)$ $r^* = KH_{k_2^*}(key^*, key, etc)$ $s^* = x^*/(r^* + x_b) \bmod q$
$tag = MAC_{key \oplus key^*}(NC_b)$	$\Rightarrow \quad tag \quad \Rightarrow$ (optional)	verify whether $tag = MAC_{key \oplus key^*}(NC_b)$

Direct Key Exchange Using a Time-Stamp (Protocol DKEUTS)		
Alice		**Bob**
$key \in_R \{0,1\}^{\ell_k}$ $x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ Get a current time-stamp TS $c = E_{k_1}(key, TS)$ $r = KH_{k_2}(key, TS, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $(key, TS) = D_{k_1}(c)$ Accept key only if $\quad TS$ is fresh and $\quad KH_{k_2}(key, TS) = r$
(k_1^*, k_2^*) $\quad = hash((y_b \cdot g^{r^*})^{s^* \cdot x_a} \bmod p)$ $(key^*, TS^*) = D_{k_1^*}(c^*)$ Accept key^* only if $\quad TS^*$ is fresh and $\quad KH_{k_2^*}(key^*, TS^*, key, etc) = r^*$	$\Leftarrow \quad c^*, r^*, s^* \quad \Leftarrow$	$key^* \in_R \{0,1\}^{\ell_k}$ $x^* \in_R [1, \ldots, q-1]$ $(k_1^*, k_2^*) = hash(y_a^{x^*} \bmod p)$ Get a current time-stamp TS^* $c^* = E_{k_1^*}(key^*, TS^*)$ $r^* = KH_{k_2^*}(key^*, TS^*, key, etc)$ $s^* = x^*/(r^* + x_b) \bmod q$
$tag = MAC_{key \oplus key^*}(TS)$	$\Rightarrow \quad tag \quad \Rightarrow$ (optional)	verify whether $tag = MAC_{key \oplus key^*}(TS)$

Table 7: Direct Key Material Exchange Achieving Mutual Identification

7.1 Message Compactness and Computational Cost

Every message in the key transport protocols proposed in this paper is compact and can be carried by a short packet such as a single ATM cell. In terms of computational cost, it takes one modular exponentiation on Alice's side, and two modular exponentiations on Bob's side which can be reduced to 1.17 exponentiations (on average) when Shamir's method for fast evaluation of the product of several exponentials with the same modulo (see [12]). As for pre-computation, the exponentiation by Alice, $y_b^x \bmod p$, can be done prior to the start of an execution of a protocol, only if Alice knows beforehand that she is going to communicate with Bob at a later time. In what follows we compare our protocols with two different sets of session key establishment protocols which serve as representatives employing the signature-then-encryption approach.

7.1.1 Comparison with ATM Forum Proposals

First we consider a proposed standard related to security in ATM. The current version of Phase I ATM Security Specification [7, 26] contains two key material exchange protocols. One involves three and the other two moves or flows of messages (see Sections 6.1.1 and 6.1.2 of [7]). These two protocols have been largely based on X.509 [15]. To describe the protocols defined in the Specification [7], we use the following symbols and abbreviations which are essentially the same as those defined the document.

1. ID_a is the (distinguished) name of Alice.
2. T_a is a time-stamp generated by Alice, consisting of a 4-byte coordinated universal time and a 4-byte sequential number.
3. R_a a nonce generated by Alice.
4. $Enc_{K_b}(\cdot)$ denotes encryption by Alice using either a secret key or a public key algorithm.
5. $ConfPar_a$ contains key materials from Alice.
6. $Sig_{K_a}(\cdot)$ denotes signature generation by Alice.
7. $cert_a$ denotes Alice's public key certificate and is used in the three-way protocol.
8. $SecNeg_a$ carries information on types of security services to be provided, algorithm and protocol options available and parameters requested for a connection. It is used only in the three-way protocol.
9. *SecOpt* is generated by Alice, carrying information similar to that contained in *SecOpt*, although it is used only in the two-way protocol.
10. ID_b, T_b, R_b, $Enc_{K_a}(\cdot)$, $ConfPar_b$, $Sig_{K_b}(\cdot)$, $SecNeg_b$ and $cert_b$ are all associated with Bob and defined similarly.
11. $\{X\}$ indicates that X is optional.

With the aid of the above symbols, we summarize in Table 8 the two protocols proposed in Phase I ATM Security Specification.

Three-Way Key Material Exchange Protocol		
Alice (Initiator)		**Bob (Respondent)**
$\Rightarrow$	$ID_a, \{ID_b\}, R_a, SecNeg_a, \{Cert_a\}$	$\Rightarrow$
$\Leftarrow$	$ID_a, ID_b, SecNeg_b, \{Cert_b\},$ $\{R_a, R_b, \{Enc_{K_a}(ConfPar_b)\},$ $Sig_{K_b}(hash(ID_a, ID_b, R_a, R_b, SecNeg_a, SecNeg_b, \{ConfPar_b\}))\}$	$\Leftarrow$
$\Rightarrow$	$\{ID_a, ID_b, R_b, \{Enc_{K_b}(ConfPar_a)\},$ $Sig_{K_a}(hash(ID_a, ID_b, R_b, \{ConfPar_a\}))\}$	$\Rightarrow$

Two-Way Key Material Exchange Protocol		
Alice (Initiator)		**Bob (Respondent)**
$\Rightarrow$	$ID_a, ID_b, SecOpt, \{T_a, R_a, \{Enc_{K_b}(ConfPar_a)\},$ $Sig_{K_a}(Hash(ID_a, ID_b, T_a, R_a, SecOpt, \{ConfPar_a\}))\}$	$\Rightarrow$
$\Leftarrow$	$\{ID_a, ID_b, R_a, \{Enc_{K_a}(ConfPar_b)\},$ $Sig_{K_b}(Hash(ID_a, ID_b, R_a, \{ConfPar_b\}))\}$	$\Leftarrow$

Table 8: Key Material Exchange Protocols Proposed by ATM Forum

It is stated in Phase I ATM Security Specification that the two key material exchange protocols can be implemented either in secret key (symmetric) cryptography or public key (asymmetric) cryptography. What we are interested in the present work is the latter, namely, the case when the two protocols are implemented in public key cryptography. Leaving out some of the technical details which are not directly relevant to our analysis, it becomes clear that both protocols follow the traditional signature-then-encryption approach. Furthermore, we can see that the three-way key material exchange protocol is based on nonces, while the two-way protocol is based on a time-stamp (together with a nonce). Thus the three-way protocol achieves similar goals to those by our protocol DKEPUN described in Table 7, and the two-way protocol achieves similar goals to those by our protocol DKEPUTS described in the same table. As is expected, our signcryption-based protocols DKEPUN and DKEPUTS are significantly more efficient than their respective counterparts proposed by the ATM Forum, both in terms of computational cost and message overhead. A detailed comparison can be easily worked out by the use of Tables 3 and 4.

7.1.2 Comparison with Beller-Yacobi Protocol

The next protocol we examine is an efficient proposal by Beller and Yacobi [3]. Their protocol is briefly summarized in Table 9, using notations consistent with those for signcryption schemes. As is the case for our proposals based on signcryption, here it is assumed too that public key certificates have already been transferred prior to an execution of the protocol. In Beller-Yacobi protocol, Alice uses ElGamal signature scheme to sign a message, and cubic RSA to encrypt the message before delivering it to Bob. Bob holds the matching cubic RSA decryption key and hence can extract the message. The number of modular exponentiations done by Alice is one (for signature generation), and by Bob is four (one for decrypting cubic RSA and three for verifying Alice's digital signature). Shamir's technique for fast evaluation of the product of several exponentials with the same modulo can also be used to speed-up the verification of ElGamal signature by Bob. More specifically, the cost for computing the product of modulo three exponentiations on Bob's side can be reduced to 1.25 modulo exponentiations on average. It is important to note that since the decryption

operation for the cubic RSA on Bob's side involves an exponentiation with a full size exponent, it can be very time-consuming, especially when the RSA composite is large. Table 10 indicates that our protocols are indeed advantageous compared to Beller-Yacobi protocol.

Alice		**Bob**
$K \in_R \{0,1\}^{\ell_k}$ $c_1 = K^3 \bmod n_B$	$\Rightarrow c_1 \Rightarrow$	Extract K from c_1 by using the decryption key associated with the RSA composite n_B
Decrypt c_2 and verify the format of the message	$\Leftarrow c_2 \Leftarrow$	Choose a random m $c_2 = E_K(m, 0^t)$
Compute ElGamal signature (v, w) on (m, etc) $c_3 = E_K(v, w, etc)$	$\Rightarrow c_3 \Rightarrow$	Decrypt c_3 and verify (v, w)

Table 9: Beller-Yacobi Authenticated Key Transport Protocol

Protocols	Comp. Cost # of exp.	Pre-Comp. by Alice	Longest Message (typical example)
Beller-Yacobi	$1 + 2.25^*$	Yes	$\geqq \lvert n_B \rvert$ bits ($\geqq 512$ bits)
DKTUN & DKTUTS	$1 + 1.17$	Yes$^+$	$\leqq 384$ bits (< 384 bits)
IKTUN	$1 + 1.17$	Yes$^+$	< 384 bits (240 bits)
IKTUTS	$1 + 1.17$	Yes$^+$	< 384 bits (280 bits)

* Including an RSA decryption with a full size exponent.
$^+$ Only when Alice knows whom to communicate with.

Table 10: Comparison with Beller-Yacobi Protocol

7.2 Remarks on Forward Secrecy

Recall that a key establishment protocol is said to offer forward secrecy with respect to a participant if compromise of the participant's long term secret key does not result in compromise of past session keys. Clearly a key establishment protocol based on a shared static key between two participants cannot offer forward secrecy.

Among protocols that are based on public key cryptography and offer forward secrecy with respect to both participants are those derived from the Diffie-Hellman key establishment protocol (see for example protocols proposed in [11, 14]). Adding to these is Beller-Yacobi protocol [3] which offers forward secrecy with respect to Alice the sender (but not with respect to Bob the receiver). In contrast, the signcryption based key transport protocols proposed in this paper do not offer forward secrecy with respect to either participant.

However, it is our view that one cannot categorically claim that a key establishment protocol with forward secrecy is better than one without. Rather one should take into

account the additional computational and communication overhead involved in providing forward secrecy.

There are basically two approaches that may be employed in containing potential damages due to compromise of a long term secret key. The first is to design a key establishment protocol that offers forward secrecy and hence can tolerate compromise of the key. The second is to find a way to make the key less compromiseable. As will be shown immediately, the second approach seems far more economical than the first one in terms of extra computational cost involved.

Before proceeding to a discussion on how to protect a participant's long term secret key from being compromised, we note that there are mainly two possible threats to the long term secret key: accidental loss and, more serious, theft. It turns out that both threats can be effectively thwarted via such means as secret sharing, either in a mathematical [27] or physical sense.

To illustrate how simple and effective a secret sharing method is against the theft and accidental loss of a long term secret key, we take a look at Alice's long term secret key x_a. What Alice can do is to choose a random number $x_{a,1}$, calculate $x_{a,2} = x_a \oplus x_{a,1}$, and then store $x_{a,1}$ and $x_{a,2}$ in two different secure locations. These secure locations can be logically separate secure compartments in Alice's computer system, two physical devices (say, one is a tamper-resistant smart card, the other a PC with a lock), or a combination of logically and physically secure facilities. One can see that Alice can readily recover x_a from $x_{a,1}$ and $x_{a,2}$ by computing $x_a = x_{a,1} \oplus x_{a,2}$, and the extra computational cost to be invested by Alice is negligible. However, for an attacker or intruder to successfully steal x_a, he has to break into both secure locations, a task that would be twice as hard as breaking into one of them.

The above method is called a 2 out of 2, or $(2, 2)$ threshold secret sharing scheme. It can be extended to $(3, 3)$, $(4, 4)$ and so on. More generally, Alice can use a t out of n, or (t, n) threshold secret sharing scheme, where $n \geqq t$, in safeguarding her long term secret key x_a. An example of (t, n) threshold secret sharing schemes is Shamir's scheme based on polynomial interpolation on a finite field [27]. The computational cost involved in a (t, n) secret sharing scheme is marginal when compared with an exponentiation modulo a large integer. No information, in an information-theoretic sense, on a long term secret key dispersed in a (t, n) threshold scheme is leaked to an attacker even if he has managed to break into up to $t - 1$ of the secure locations. An added benefit is that Alice can still reconstruct x_a even when up to $n - t$ secure locations are un-recoverably damaged.

8 Multicast Conference Key Establishment

The two protocols for direct transport of key materials described in Section 6.2 can be extended to conference key establishment where Alice wishes to establish a common session key with t recipients R_1, R_2, ..., R_t. Such a protocol is very useful in multicast communications. A major difference between a single recipient protocol and a multiple recipient one, both based on signcryption, lies in the length of messages. As shown in previous sections, messages in a key establishment protocol for a single recipient are all compact and can be accommodated in small data packets such as ATM cells. With a protocol for multiple recipients, some messages may be too long to fit in a single ATM cell. Therefore one of our design goals will be use as a small number of cells as possible in transporting key materials.

We assume that each recipient R_i has a unique identifier ID_i, and that the private key

of R_i is $x_i \in_R [1, \ldots, q-1]$, and his matching public key is $y_i = g^{x_i} \bmod p$. Two basic multicast conference key transport protocols are shown in Table 11. The main part of these protocols is a multicast message containing key materials from Alice to all t recipients R_1, R_2, ..., R_t. The data format for the multicast message is adapted from a signcryption scheme for multiple recipients proposed in [28, 30].

An interesting property of these protocols is that after a successful run of the protocols, all recipients are assured that session keys in their hands are consistent. In other words, all recipients recover an identical session key from their copies of a multicast message, which would prevent a particular recipient from being excluded from the multicast group by a dishonest Alice. This property on session key consistency is achieved through the use of two techniques: (1) *key*, a session key, is encrypted *together with the hashed value* $h = KH_k(key, TQ, etc)$, namely $c = E_k(key, h)$, where TQ can be either a time-stamp (TS) or a nonce (NC) and *etc* may contain Alice's public key certificate, a multicast group identifier and other data. (2) *key* and k are both involved in the generation of r_i and s_i through $r_i = KH_{k_{i,2}}(h, etc_i)$, where etc_i may contain R_i's public key certificate and other data.

As for computational load, it takes t modular exponentiations on Alice's side, and on average 1.17 modular exponentiations one each recipient R_i's side. Note that with the second protocol that relies a time-stamp, the relatively computationally expensive t modular exponentiations are carried out prior to fetching a time-stamp. This would help reduce the chance of obtaining an inaccurate time-stamp.

In some applications, however, there might be two potential problems with these basic protocols. The first problem is that multicasting tag_i (and NC_i) by each R_i may flood a network in a situation where the number of recipients is large and/or the network is already too congested. And the second problem is that the process of verifying all confirmation tags may pose a computational burden on a recipient R_i.

These two potential problems may be solved using a randomization technique.

1. With the first protocol that involves nonces, instead of always generating and multicasting NC_i, each R_i may first flip a (biased or unbiased) coin and then according the outcome of the coin-flipping, decide whether or not generating and multicasting NC_i.

2. With both protocols, R_i may decide, in a probabilistic fashion, whether or not generating and multicasting tag_i.

3. With both protocols, Alice and each R_i may randomly choose a subset of the key confirmation tags received for verification, rather than going through the process of checking every tag arrived.

Clearly there is a trade-off between the number of confirmation tags generated/verified and the level of confidence in key confirmation.

Finally, we examine methods for data transfer for the multicast key establishment protocols.

1. On some networks, the identifier ID_i of R_i may need not to be transferred with NC_i or tag_i, as it may be implicitly included in a packet that carries NC_i or tag_i.

2. Since NC_i, $i = 1, \ldots, t$, are generated independently of one another, they should be multicast independently too. The same can be said with tag_i, $i = 1, \ldots, t$.

3. There are two methods to multicast $(c, c_1, r_1, s_1, \ldots, c_t, r_t, s_t)$:

   (a) Packing these data items into as a small number of data packets as possible,

   (b) Placing (c, c_1, r_1, s_1) into the first data packet, (c, c_2, r_2, s_2) into the second data packet, and so on.

   An advantage of the first method for multicast $(c, c_1, r_1, s_1, \ldots, c_t, r_t, s_t)$ is that it consumes the smallest possible data packets and hence would be very efficient in terms of communication overhead. In contrast, although the second method may require a larger number of data packets than the first, it has an advantage in terms of its simplicity. We note, however, that with the second method, each recipient still has to check all the data packets he receives to ensure that the first parts of the packets are identical (to c).

   We also note that marking or labeling may be used in order for each R_i to correctly extract (c, c_i, r_i, s_i).

9 Conclusion

We have presented a number of compact and authenticated key establishment protocols. These are the first protocols based on public key cryptography whose messages can all be carried in very small data packets such as single ATM cells. We have further extended the protocols to conference key establishment where a multiple number of recipients are involved. Our protocols have all been built on the signcryption primitive which fulfills both the functions of signature and encryption in an efficient way. A detailed analysis and comparison has shown that the overall computational cost of these protocols is significantly smaller than all other currently known protocols that are based on public key cryptography.

Acknowledgment

We would like to thank anonymous reviewers for INFOCOM'98 for their helpful comments. Part of this work was completed while the first author was on sabbatical leave at the University of Tokyo.

References

[1] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology - CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Berlin, New York, Tokyo, 1996. Springer-Verlag.

[2] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology - CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Berlin, New York, Tokyo, 1993. Springer-Verlag.

[3] M. Beller and Y. Yacobi. Fully-fledged two-way public key authentication and key agreement for low cost terminals. *Electronic Letters*, 30:999–1001, 1993.

Conference Key Establishment Using a Nonce

Alice and each R_i, $i = 1, \ldots, t$: $NC = NC_1 \oplus \cdots \oplus NC_t$	$\nwarrow \leftarrow \swarrow$ $\begin{pmatrix} ID_1, NC_1 \\ \vdots \\ ID_t, NC_t \end{pmatrix}$ $\swarrow \leftarrow \nwarrow$	**Each R_i, $i = 1, \ldots, t$:** $NC_i \in_R \{0,1\}^{\ell_n}$
Alice: $key \in_R \{0,1\}^{\ell_1}$ $k \in_R \{0,1\}^{\ell_2}$ $h = KH_k(key, NC, etc)$ $c = E_k(key, h)$ for each $i = 1, \ldots, t$: $v_i \in_R [1, \ldots, q-1]$ $(k_{i,1}, k_{i,2}) = hash(y_i^{v_i} \bmod p)$ $c_i = E_{k_{i,1}}(k)$ $r_i = KH_{k_{i,2}}(h, etc_i)$ $s_i = v_i/(r_i + x_a) \bmod q$	$\rightarrow$ $\begin{pmatrix} c \\ c_1, r_1, s_1 \\ \vdots \\ c_t, r_t, s_t \end{pmatrix}$ $\nearrow \rightarrow \searrow$	**Each R_i:** Find out (c, c_i, r_i, s_i) $(k_{i,1}, k_{i,2})$ $= hash((y_a \cdot g^{r_i})^{s_i \cdot x_i} \bmod p)$ $k = D_{k_{i,1}}(c_i)$ $(key, h) = D_k(c)$ Accept key only if $KH_k(key, NC, etc) = h$ and $KH_{k_{i,2}}(h, etc_i) = r_i$
Alice verifies all $tag_1, \ldots, tag_t$; **Each R_i** verifies tag_j for each j with $j \neq i$ and $1 \leqq j \leqq t$.	$\nwarrow \leftarrow \swarrow$ $\begin{pmatrix} ID_1, tag_1 \\ \vdots \\ ID_t, tag_t \end{pmatrix}$ $\swarrow \leftarrow \nwarrow$ (optional)	**Each R_i:** $tag_i = MAC_{key}(NC, NC_i)$

Conference Key Establishment Using a Time-Stamp

Alice: for each $i = 1, \ldots, t$: $v_i \in_R [1, \ldots, q-1]$ $(k_{i,1}, k_{i,2}) = hash(y_i^{v_i} \bmod p)$ $key \in_R \{0,1\}^{\ell_1}$ $k \in_R \{0,1\}^{\ell_2}$ Get a current time-stamp TS $h = KH_k(key, TS, etc)$ $c = E_k(key, TS, h)$ for each $i = 1, \ldots, t$: $c_i = E_{k_{i,1}}(k)$ $r_i = KH_{k_{i,2}}(h, etc_i)$ $s_i = v_i/(r_i + x_a) \bmod q$	$\rightarrow$ $\begin{pmatrix} c \\ c_1, r_1, s_1 \\ \vdots \\ c_t, r_t, s_t \end{pmatrix}$ $\nearrow \rightarrow \searrow$	**Each R_i, $i = 1, \ldots, t$:** Find out (c, c_i, r_i, s_i) $(k_{i,1}, k_{i,2})$ $= hash((y_a \cdot g^{r_i})^{s_i \cdot x_i} \bmod p)$ $k = D_{k_{i,1}}(c_i)$ $(key, TS, h) = D_k(c)$ Accept key only if TS is fresh, $KH_k(key, TS, etc) = h$ and $KH_{k_{i,2}}(h, etc_i) = r_i$
Alice verifies all $tag_1, \ldots, tag_t$; **Each R_i** verifies tag_j for each j with $j \neq i$ and $1 \leqq j \leqq t$.	$\nwarrow \leftarrow \swarrow$ $\begin{pmatrix} ID_1, tag_1 \\ \vdots \\ ID_t, tag_t \end{pmatrix}$ $\swarrow \leftarrow \nwarrow$ (optional)	**Each R_i:** $tag_i = MAC_{key}(TS, ID_i)$

Table 11: Key Establishment for Multicasting (Basic Protocol)

[4] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of efficient provably secure two-way authentication protocols. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 44–61, Berlin, New York, Tokyo, 1992. Springer-Verlag.

[5] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. The KryptoKnight family of authentication and key distribution protocols. *IEEE/ACM Transactions on Networking*, 1995.

[6] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[7] The ATM Forum Technical Committee. Phase I ATM security specification (draft), July 1997. ATM Forum BTD-SECURITY-01.03.

[8] The ATM Forum Technical Committee. Security framework for ATM networks (draft), April 1997. ATM Forum BTD-FRWK-01.00.

[9] Y. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, 1994.

[10] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):472–492, 1976.

[11] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and authenticated key exchange. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[12] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[13] L. Gong. Efficient network authentication protocols: Lower bounds and optimal implementations. *Distributed Computing*, 9(3):131–145, 1995.

[14] D. Harkins and D. Carrel. The resolution of ISAKMP with Oakley, March 1998. Internet-draft (draft-ietf-ipsec-isakmp-oakley-07.txt).

[15] ITU. Information technology - open systems interconnection - the directory: Authentication framework. Recommendation X.509, International Telecommunications Union, 1993.

[16] P. Janson, G. Tsudik, and M. Yung. Scalability and flexibility in authentication services: The KryptoKnight approach. In *Proceedings of INFOCOM'97*. IEEE, 1997.

[17] D. Johnson and S. Matyas. Asymmetric encryption: Evolution and enhancements. *CryptoBytes*, 2(1):1–6, 1996. (available at `http://www.rsa.com/`).

[18] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[19] J. Kohl and B. C. Neuman. The Kerberos network authentication services (v5). Request for Comments RFC 1510, IETF, 1993.

[20] T. Matsumoto and H. Imai. On the key predistribution systems: A practical solution to the key distribution problem. In *Advances in Cryptology - CRYPTO'87*, volume 239 of *Lecture Notes in Computer Science*, pages 185–193, Berlin, New York, Tokyo, 1987. Springer-Verlag.

[21] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[22] National Bureau of Standards. Data encryption standard. Federal Information Processing Standards Publication FIPS PUB 46, U.S. Department of Commerce, January 1977.

[23] National Institute of Standards and Technology. Digital signature standard (DSS). Federal Information Processing Standards Publication FIPS PUB 186, U.S. Department of Commerce, May 1994.

[24] National Institute of Standards and Technology. Secure hash standard. Federal Information Processing Standards Publication FIPS PUB 180-1, U.S. Department of Commerce, April 1995.

[25] H. K. Orman. The Oakley key determination protocol, July 1997. Internet-draft (draft-ietf-ipsec-oakley-02.txt).

[26] M. Peyravian and T. Tarman. Asynchronous transfer mode security. *IEEE Network*, 11(3):34–40, May/June 1997.

[27] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[28] Y. Zheng. Digital signcryption or how to achieve cost(signature & encryption) << cost(signature) + cost(encryption). In *Advances in Cryptology - CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 165–179, Berlin, New York, Tokyo, 1997. Springer-Verlag.

[29] Y. Zheng. The SPEED cipher. In *Proceedings of Financial Cryptography'97*, volume 1318 of *Lecture Notes in Computer Science*, pages 71–89, Berlin, New York, Tokyo, 1997. Springer-Verlag.

[30] Y. Zheng. Signcryption and its applications in efficient public key solutions. In *Information Security — Proceedings of 1997 Information Security Workshop (ISW'97)*, volume 1396 of *Lecture Notes in Computer Science*, pages 291–312, Berlin, New York, Tokyo, 1998. Springer-Verlag. (an invited talk).

[31] Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL - a one-way hashing algorithm with varialbe length of output. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology - AUSCRYPT'92*, volume 718 of *Lecture Notes in Computer Science*, pages 83–104, Berlin, New York, Tokyo, 1993. Springer-Verlag.