Efficient and Unforgeable Multicast Conference Key Establishment without Relying on a Key Distribution Center

Yuliang Zheng * Hideki Imai †

Abstract— 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 a cryptographic session key establishment protocol. Our earlier work shows that by using recently invented "signcryption" schemes, key establishment with the following properties can be achieved: (1) each message exchanged between two participants can be transferred in a short packet such as an ATM cell whose payload has only 384 bits, and (2) messages that carry key materials are unforgeable and non-repudiatable without the involvement of a trusted key distribution center. The main contribution of this paper is to extend our previous key establishment protocols, which have been designed for single recipients, to multicast conference key establishment in which a multiple number of recipients are involved.

Keywords— Cryptography, Key Establishment, Multicast, Network Security, Signcryption

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 used 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 connections, 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.

Organization of the rest of this paper: Section 2 presents a example implementation of signcryption that will be used in key establishment. Section 3 introduces our previous work on key establishment for a single recipient, and Section 4 shows how to extend the work to multicast conference key establishment.

* Peninsula School of Computing, Monash University, McMahons Road, Frankston, Melbourne, Victoria 3199, Australia.
URL: http://www-pscit.fcit.monash.edu.au/~yuliang/
E-mail: yuliang@mars.fcit.monash.edu.au

† Institute of Industrial Science, University of Tokyo, 22-1, Roppongi 7-chome, Minatoku, Tokyo, 106 Japan.
E-mail: imai@imailab.iis.u-tokyo.ac.jp

2 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.*

Next 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. The reader is directed to [1, 2] 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. 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.

Assume that Alice also 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 $y_b = g^{x_b} \bmod p$.

As shown in Table 1, the signcryption and unsigncryption algorithms 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 algorithm described in the left column of the table, 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.

Signcryption schemes can also be derived from

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

Table 1: An Example Implementation of Signcryption (SCS1)

ElGamal-based signature schemes built on other versions of the discrete logarithm problem such as that on elliptic curves.

3 Key Establishment for a Single Recipient

Now we are ready to describe our earlier work [3] on 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.

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. 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.500 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 $\ell_k \geq 64$, $\ell_n \geq 40$, 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.

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 2 describes two direct key transport protocols that are relevant to this work. The reader is directed to [3] for other 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, the time-stamp TS may be transmitted to Bob in clear to further improve the computational efficiency of the protocols.

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

4 Multicast Conference Key Establishment

The two protocols for direct transport of key materials described in Section 3 can be extended to conference key establishment where Alice wishes to establish a common session key with t recipients $R_1, R_2, \ldots, R_t$. 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 3. The main part of these protocols is a multicast message containing key

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 2: Direct Key Material Transport for a Single Recipient

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 [2].

An interesting property of these protocols is that after an 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.

There are 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. 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 cointossing, 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.

References

[1] 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. (The latest version of the paper is available at `http://www-pscit.fcit.monash.edu.au/~yuliang/`).

[2] Y. Zheng. Signcryption and its applications in efficient public key solutions. In *Proceedings of 1997 Information Security Workshop (ISW'97)*, Lecture Notes in Computer Science, Berlin, New York, Tokyo, 1997. Springer-Verlag.

[3] Y. Zheng and H. Imai. Secure and authenticated key establishment in a single ATM cell. Technical Report of IEICE ISEC97-24, IEICE, Tokyo, Japan, 1997.

Conference Key Establishment Using a Nonce

Alice and each R_i, $i = 1, \ldots, t$: $NC = NC_1 \oplus \cdots \oplus NC_t$	$\nwarrow$ $\leftarrow$ $\swarrow$ $\left(\begin{array}{c} ID_1, NC_1 \\ \vdots \\ ID_t, NC_t \end{array} \right)$ $\swarrow$ $\leftarrow$ $\nwarrow$	**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 $i = 1, \ldots, t$: $v_i \in_R [1, \ldots, q-1]$ $(k_{i,1}, k_{i,2}) = hash(y_i^{v_i} \bmod p)$ $c_i = E_{k_{i,1}}(k)$ $r_i = KH_{k_{i,2}}(h, etc_i)$ $s_i = v_i/(r_i + x_a) \bmod q$	$\rightarrow$ $\left(\begin{array}{c} c \\ c_1, r_1, s_1 \\ \vdots \\ c_t, r_t, s_t \end{array} \right)$ $\nearrow$ $\rightarrow$ $\searrow$	**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 $1 \leq j \leq t$.	$\nwarrow$ $\leftarrow$ $\swarrow$ $\left(\begin{array}{c} ID_1, tag_1 \\ \vdots \\ ID_t, tag_t \end{array} \right)$ $\swarrow$ $\leftarrow$ $\nwarrow$ (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 $i = 1, \ldots, t$: $c_i = E_{k_{i,1}}(k)$ $r_i = KH_{k_{i,2}}(h, etc_i)$ $s_i = v_i/(r_i + x_a) \bmod q$	$\rightarrow$ $\left(\begin{array}{c} c \\ c_1, r_1, s_1 \\ \vdots \\ c_t, r_t, s_t \end{array} \right)$ $\nearrow$ $\rightarrow$ $\searrow$	**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 $1 \leq j \leq t$.	$\nwarrow$ $\leftarrow$ $\swarrow$ $\left(\begin{array}{c} ID_1, tag_1 \\ \vdots \\ ID_t, tag_t \end{array} \right)$ $\swarrow$ $\leftarrow$ $\nwarrow$ (optional)	**Each R_i:** $tag_i = MAC_{key}(TS, ID_i)$

Table 3: Key Establishment for Multicasting (Basic Protocol)