

Using Compatible Keys for Secure Multicasting in E-Commerce

Indrakshi Ray

Indrajit Ray

Department of Computer Science

Colorado State University

601 S. Howes Street

Fort Collins, CO 80523-1873

Email: {iray, indrajit}@cs.colostate.edu

Abstract

Consider an electronic commerce (e-commerce) environment where the consumers subscribe to data objects; a server distributes these objects electronically which are then retrieved by the consumer. We propose a protocol showing how the distribution of data objects can be done in a secure and efficient manner. In our protocol, (1) consumers who have subscribed for an object can get access to the object only during the period that their subscription is valid, (2) consumers who have not subscribed to an object do not get access to the object, (3) increasing the number of consumers subscribing to an object does not deteriorate the quality of service, (4) multiple key distributions to each consumer is avoided, (5) key management is simple – each consumer just remembers one key, (6) bandwidth requirements are low, and (7) large processing overhead required for decryption is not incurred by the consumer. The protocol is based on the theory of compatible keys which we illustrate in this paper.

1. Introduction

Consider an electronic commerce (e-commerce) environment where consumers subscribe to data objects. Examples of such objects may be stock, weather, airline information etc. These objects are electronically delivered to the consumers over the Internet. The restrictions imposed on delivery are that (1) consumers who have subscribed to an object should be able to access it, (2) no one other than the subscribing consumers should have access to that object, and (3) the access should be allowed only when the subscription is valid. These objectives can be met with a simple solution: encrypt the data object with the consumer's public key and transmit this encrypted data. However, if many consumers subscribe to a data object, then this data object will be encrypted multiple times and transmitted multiple

times – this solution is clearly inefficient wasting computation and bandwidth. We explore the alternate solution of broadcasting the encrypted data and providing an access control mechanism by which only legitimate consumers can access the data. Since the consumers subscribing to the data object vary with time, the problem we are addressing is of secure multicasting in a dynamic environment.

Secure broadcasting/multicasting at the application level has received some attention in the past few years [4, 7, 9, 10]. However, these fail to satisfy one or more of the following requirements: (1) The protocol should be scalable; increasing the number of subscribers to an object should not deteriorate the quality of service. (2) Key management should be simple. Consumers should not have to keep track of multiple keys. Ideally, each consumer should remember only one key. Each server should not have to keep track of a large number of keys. (3) Key distributions to the consumers should be kept at the minimum - ideally one for each consumer. Key distribution to the consumers incurs a significant overhead both in terms of bandwidth and time. Reducing key distribution will result in a more efficient protocol. (4) Consumers should not incur significant processing overhead for decryption. We propose a solution that satisfies all the above mentioned requirements.

In our mechanism we have the consumers who subscribe to various data objects. A *subscription* allows a consumer to access one or more data objects for specific period(s) of time. A consumer cannot access the data object beyond the specified period. A consumer is allowed to change or cancel his subscription at any point of time. The consumer interacts with the *server* when he wishes to add/modify/cancel his subscription. The server has access to a large repository of data objects. At each broadcast session, the server determines which objects must be broadcast and who are the recipients of these objects. Based on this information, the server generates the key required to encrypt each object. This encrypted object is a part of the message that is broadcast by the server.

The encryption is based on the theory of compatible keys described in Section 3. Briefly, it is as follows. For each consumer C_i that subscribes, the server generates the key pair $\langle K_i, K_i^{-1} \rangle$. This key pair is mathematically related to all existing key pairs $\langle K_j, K_j^{-1} \rangle$ belonging to consumers C_j s. The consumer C_i receives the key K_i^{-1} and uses it to decrypt any object that he has subscribed to.

To illustrate our encryption mechanism we give an example. Initially let C_1 be the only consumer who has subscribed to a data object m . The server encrypts the data object m with the key K_1 (denoted by $[m, K_1]$), and broadcasts this encrypted object. C_1 uses his decryption key K_1^{-1} to retrieve m . Thus, a consumer has to store only one decryption key to retrieve the data objects he has subscribed to. Also, note that multiple decryptions are not necessary to obtain the object.

Now suppose another consumer C_3 subscribes to the data object m . The server prepares key pair $\langle K_3, K_3^{-1} \rangle$ for C_3 such that K_3 is compatible with K_1 (we define compatibility of keys in Section 3). The server now encrypts the data object with the key $K_1 \times K_3$ (we define \times operator in Section 3), denoted by $[m, K_1 \times K_3]$, and broadcasts it. The key $K_1 \times K_3$ is such that consumer C_1 is able to obtain m by decrypting the broadcast message using K_1^{-1} and C_3 is able to obtain m by decrypting the broadcast message using K_3^{-1} . Thus, when the number of consumers subscribing to a data object increases, the only change necessary is the way in which the data is encrypted. The size of the broadcast message is not changed. Thus, our protocol is scalable.

At some later time, the group of consumers subscribing to m changes: consumer C_3 no longer subscribes to m and consumers C_2 and C_4 have started subscribing to m . The server as before generates key pairs $\langle K_2, K_2^{-1} \rangle$ for C_2 and $\langle K_4, K_4^{-1} \rangle$ for C_4 . To deliver the data, the server broadcasts the data object m encrypted with key $K_1 \times K_2 \times K_4$. From the encrypted object, $[m, K_1 \times K_2 \times K_4]$, only consumers C_1 , C_2 and C_4 can retrieve m . Note that, key distribution is not necessary when the group (subscribing for a particular object) changes. Only the consumers who have been added/deleted from the group are aware of the change – the change is transparent to the existing consumers.

The rest of the paper is organized as follows. Section 2 describes some related work in this area. Section 3 describes the theory of compatible keys on which our protocol is based. Section 4 describes the detailed mechanism. Finally, section 5 concludes the paper.

2. Related Work

One of the earlier works in secure broadcasting is due to Gopal and Jaffe [9]. The authors proposed a point-to-point approach to broadcasting. For each consumer, the server makes an encryption of the data object, and broadcasts this.

Suppose there are N consumers C_1, C_2, \dots, C_N having public keys P_1, P_2, \dots, P_N respectively. These N consumers subscribe to the data item D_i . Then the broadcast message will consist of the concatenation of $[D_i, P_1], [D_i, P_2], [D_i, P_3], [D_i, P_4], \dots [D_i, P_N]$ where $[D_i, P_j]$ refers to the encryption of D_i with the public key P_j . Thus, the same message is encrypted multiple times and broadcast – not efficient.

Chiou and Chen [7] proposed a session key approach to broadcasting. In this approach each broadcast is considered a session. The session keys are used for one session and then discarded. Corresponding to each data item D_i there is a pair of encryption/decryption keys ($SessEncKey_i, SessDecKey_i$). The server encrypts the data item D_i with the $SessEncKey_i$ and broadcasts it. The consumers who subscribed to the data item D_i need the $SessDecKey_i$ in order to decrypt the data. For each consumer C_j subscribing to the data item D_i , the server encrypts $SessDecKey_i$ with the public key of the consumer P_j . The encrypted $SessDecKey_i$ for all the consumers who have subscribed to data item D_i are concatenated together with the data item D_i and broadcast. In short, if consumers C_1, C_2, \dots, C_N subscribe to the data item D_i , the following message will be broadcast: concatenation of $[D_i, SessEncKey_i], [SessDecKey_i, P_1], [SessDecKey_i, P_2], \dots, [SessDecKey_i, P_N]$. Clearly, the length of the message will depend on the number of consumers subscribing to the data item. Note that a consumer who has subscribed to multiple data items will receive multiple decryption keys for deciphering all these items. The consumer thus has to manage multiple keys. However, key management is somewhat simplified due to the fact that session keys are used for one session only and are discarded after use. Key distribution is a major problem with this approach: the session keys must be distributed to all subscribers at every broadcast.

Rather than broadcast session keys at every session, an alternate approach using group keys is proposed by Ingemarsson et al. [10]. All consumers subscribing to a particular data item form a group. Associated with each group there is a group key. This group key is broadcast and used until some consumer leaves the group. If a consumer leaves the group, a new group key must be generated and distributed to the existing group members. The data item is encrypted with the group key and broadcast. If this is a new group key, then it is securely transmitted (by encrypting it with the public key of each group member) to all the members of the group. The different encryptions of the group key are concatenated with the encrypted data item and then broadcast. For the case where the consumers C_1, C_2, \dots, C_N have subscribed to a data item D_i , the following encryptions are concatenated and then broadcasted: $[D_i, G_x], [G_x, P_1], [G_x, P_2], \dots, [G_x, P_N]$ where G_x is the group key with which D_i is encrypted, P_j is the public key of consumer C_j , and $[a, b]$ denotes a encrypted with key b . Note

that a consumer who subscribes to multiple items, will have to manage multiple group keys. Since group keys are used multiple times, managing and storing multiple group keys is non-trivial. Key distribution is less expensive than the session key approach – it is necessary only when a consumer leaves the group. In all the above approaches, the size of the data that is broadcast increases as the number of consumer increases.

Celik and Datta [4] improve upon the group key approach. The time continuum is divided into discrete epochs. Clients can subscribe to data items for one or more epochs. The actual duration of an epoch is based on subscription patterns. Consumers are assigned to the same group if they subscribe to the same data items and their subscription expires at the same epoch. This obviates the need for redistributing new keys to existing group members when consumers leave the group. A group key is given to the consumer when he subscribes for one or more items. Thus, when the data item is being broadcast there is no need to send the group key. The length of the broadcast is therefore independent of the number of consumers. Now if the consumer subscribes for all data items together, he has to manage only one group key. But in real world, it is possible that a consumer subscribes to one data item, and later on decides to subscribe to a second item, and so on. In such a scenario the consumer may have to manage multiple group keys. The server may also have to manage a large number of keys. Suppose the time continuum is divided into H epochs. Then for any one item there can be $H!$ possible subscription patterns. Suppose there are a total of d items from which the consumers can choose. Number of ways consumers can choose subsets of d items is $(2^d - 1)$. So the maximum number of groups is $H! * (2^d - 1)$. Thus, in the worst case, the server has to keep track of a very large number of keys. Moreover, the authors do not address what happens if a consumer decides to terminate his subscription before the end date – this scenario might arise in a real world situation. In such a case it appears that the proposed scheme will not work.

Other works in secure multicasting include the problem of key distribution [1, 8, 11, 15] and digital signatures [2, 3, 5, 6] among the participants in a group. But most of these work do not address dynamic groups.

3. Theory of Compatible Keys

Before presenting our protocol we establish the theory of compatible keys on which the protocol is based.

Definition 1 The set of messages M is the set of non-negative integers m that are less than an upper bound N , i.e.

$$M = \{m | 0 \leq m < N\} \quad (1)$$

Definition 2 Given an integer a and a positive integer N , the following relationship holds,

$$a = qN + r \text{ where } 0 \leq r < N \text{ and } q = \lfloor a/N \rfloor \quad (2)$$

where $\lfloor x \rfloor$ denotes the largest integer less than equal to x . The value q is referred to as the *quotient* and r is referred to as the *remainder*. The remainder r , denoted $a \bmod N$, is also referred to as the *least positive residue* of $a \bmod N$.

Definition 3 For positive integers a , b and N , we say a is *equivalent* to b , modulo N , denoted by $a \equiv b \pmod{N}$, if $a \bmod N = b \bmod N$.

Definition 4 For positive integers a , x , n and $n > 1$, if $\gcd(a, n) = 1$ and $a \cdot x \equiv 1 \pmod{n}$, then x is referred to as the *multiplicative inverse* of a modulo n .

Definition 5 Two integers a , b are said to be *relatively prime* if their only common divisor is 1, that is, $\gcd(a, b) = 1$.

Definition 6 The integers n_1, n_2, \dots, n_k are said to be *pairwise relatively prime*, if $\gcd(n_i, n_j) = 1$ for $i \neq j$.

Definition 7 The Euler's totient function $\phi(N)$ is defined as the number of integers that are less than N and relatively prime to N . Below we give some properties of totient functions that we need in this paper.

1. $\phi(N) = N - 1$ if N is prime.
2. $\phi(N) = \phi(N_1)\phi(N_2) \dots \phi(N_k)$ if $N = N_1N_2 \dots N_k$ and N_1, N_2, \dots, N_k are pairwise relatively prime.

Theorem 1 Euler's theorem states that for every a and N that are relatively prime,

$$a^{\phi(N)} \equiv 1 \pmod{N}$$

Proof: We omit the proof of Euler's theorem and refer the interested reader to any book on number theory [13] or cryptography [14].

Corollary 1 If $0 < m < N$ and $N = N_1N_2 \dots N_k$ and N_1, N_2, \dots, N_k are primes, then $m^{x\phi(N)+1} \equiv m \pmod{N}$ where x is an integer.

Definition 8 A key K is defined to be the ordered pair $\langle e, N \rangle$, where N is a product of distinct primes, $N \geq M$ and e is relatively prime to $\phi(N)$; e is the *exponent* and N is the *base* of the key K .

Definition 9 The *encryption* of a message m with the key $K = \langle e, N \rangle$, denoted as $[m, K]$, is defined as

$$[m, \langle e, N \rangle] = m^e \pmod{N} \quad (3)$$

Definition 10 The *inverse* of a key $K = \langle e, N \rangle$, denoted by K^{-1} , is an ordered pair $\langle d, N \rangle$, satisfying $ed \equiv 1 \pmod{\phi(N)}$.

Theorem 2 For any message m .

$$[[m, K], K^{-1}] = [[m, K^{-1}], K] = m \quad (4)$$

where $K = \langle e, N \rangle$ and $K^{-1} = \langle d, N \rangle$.

Proof: We first show that

$$\begin{aligned} & [[m, K], K^{-1}] = m \\ LHS &= \begin{aligned} & [[m, K], K^{-1}] \\ &= [m^e \bmod N, K^{-1}] \quad (\text{Def. 9}) \\ &= (m^e \bmod N)^d \bmod N \quad (\text{Defs. 9, 10}) \\ &= m^{ed} \bmod N \quad (\text{laws of mod. arith.}) \\ &= m^{(x\phi(N)+1)} \bmod N \quad (\text{Defs. 2, 10,} \\ &\quad \text{since } ed = x\phi(N) + 1) \\ &= m \bmod N \quad (\text{Corollary 1}) \\ &= m \quad (\text{since } m < N; \text{ Def. 1}) \\ &= RHS \end{aligned} \end{aligned}$$

By symmetry $[[m, K^{-1}], K] = m$.

Corollary 2 An encryption, $[m, K]$, is one-to-one if it satisfies the relation

$$[[m, K], K^{-1}] = [[m, K^{-1}], K] = m$$

Definition 11 Two keys $K_1 = \langle e_1, N_1 \rangle$ and $K_2 = \langle e_2, N_2 \rangle$ are said to be *compatible* if $e_1 = e_2$ and N_1 and N_2 are relatively prime.

Definition 12 If two keys $K_1 = \langle e, N_1 \rangle$ and $K_2 = \langle e, N_2 \rangle$ are compatible, then the *product* key, $K_1 \times K_2$, is defined as $\langle e, N_1 N_2 \rangle$.

Lemma 1 For positive integers a, N_1 and N_2 ,

$$(a \bmod N_1 N_2) \equiv a \bmod N_1$$

Proof: Let $a = N_1 N_2 x + N_1 y + z$, where x, y and z are integers.

$$\begin{aligned} LHS &= (a \bmod N_1 N_2) \bmod N_1 \\ &= \left(N_1 N_2 x + N_1 y + z - \left\lfloor \frac{N_1 N_2 x + N_1 y + z}{N_1 N_2} \right\rfloor \times N_1 N_2 \right) \bmod N_1 \\ &= (N_1 y + z) \bmod N_1 \\ &= z \\ RHS &= (a \bmod N_1) \\ &= (N_1 N_2 x + N_1 y + z) \bmod N_1 \\ &= z \end{aligned}$$

Hence the proof.

Theorem 3 For any two messages m and \hat{m} , such that $m, \hat{m} < N_1, N_2$,

$$[m, K_1 \times K_2] \equiv [\hat{m}, K_1] \pmod{N_1} \text{ iff } m = \hat{m} \quad (5)$$

$$[m, K_1 \times K_2] \equiv [\hat{m}, K_2] \pmod{N_2} \text{ iff } m = \hat{m} \quad (6)$$

where K_1 is the key $\langle e, N_1 \rangle$, K_2 is the key $\langle e, N_2 \rangle$ and $K_1 \times K_2$ is the product key $\langle e, N_1 N_2 \rangle$.

Proof: The proof for (6) is the same as that for (5). We just consider the proof for (5).

[If part] Given $m = \hat{m}$, we have to prove that $[m, K_1 \times K_2] \equiv [\hat{m}, K_1] \pmod{N_1}$, that is,

$$[m, K_1 \times K_2] \pmod{N_1} = [\hat{m}, K_1] \pmod{N_1}$$

$$\begin{aligned} LHS &= [m, K_1 \times K_2] \pmod{N_1} \\ &= (m^e \bmod N_1 N_2) \pmod{N_1} \quad (\text{Defs. 9, 12}) \\ &= m^e \bmod N_1 \quad (\text{put } m^e \text{ for } a \text{ in lemma 1}) \end{aligned}$$

$$\begin{aligned} RHS &= [\hat{m}^e \bmod N_1] \pmod{N_1} \\ &= \hat{m}^e \bmod N_1 \quad (\text{idempotency of mod op.}) \\ &= m^e \bmod N_1 \quad (\text{since } m = \hat{m}, \text{ given}) \end{aligned}$$

[Only If part] Given $[m, K_1 \times K_2] \equiv [\hat{m}, K_1] \pmod{N_1}$, we have to prove $m = \hat{m}$

$$\begin{aligned} & [m, K_1 \times K_2] \equiv [\hat{m}, K_1] \pmod{N_1} \\ & \text{or } [m, K_1 \times K_2] \pmod{N_1} = [\hat{m}, K_1] \pmod{N_1} \quad (\text{Def. 3}) \\ & \text{or } (m^e \bmod N_1 N_2) = (\hat{m}^e \bmod N_1) \quad (\text{Defs. 9, 12}) \\ & \text{or } m^e \bmod N_1 = (\hat{m}^e \bmod N_1) \pmod{N_1} \quad (\text{lemma 1}) \\ & \text{or } [m, \langle e, N_1 \rangle] = [\hat{m}, \langle e, N_1 \rangle] \quad (\text{Def. 9}) \\ & \text{or } m = \hat{m} \quad (\text{encryption is one-to-one}) \end{aligned}$$

The more general case for the above theorem is as follows.

$$[m, K_1 \times K_2 \dots K_p] \equiv [\hat{m}, K_1] \pmod{N_1} \text{ iff } m = \hat{m} \quad (7)$$

$$[m, K_1 \times K_2 \dots K_p] \equiv [\hat{m}, K_1] \pmod{N_2} \text{ iff } m = \hat{m} \quad (8)$$

⋮

$$[m, K_1 \times K_2 \dots K_p] \equiv [\hat{m}, K_1] \pmod{N_p} \text{ iff } m = \hat{m} \quad (9)$$

where $K_1 = \langle e, N_1 \rangle$, $K_2 = \langle e, N_2 \rangle$, etc. and the product key $K_1 \times K_2 \dots K_p$ is equal to $\langle e, N_1 N_2 \dots, N_p \rangle$.

4. The Detailed Mechanism

Our protocol proceeds in phases. First a consumer registers with the server. This is followed by subscription when a consumer subscribes to one or more objects. During each

broadcast transmission, the server checks which data objects must be transmitted. The server then appropriately encrypts the data objects, concatenates these encrypted data objects into one message and broadcasts it. The consumer tunes in at the subscription times and listens to the broadcast message. The consumer then decrypts the broadcast message to obtain the data object(s) to which he has subscribed.

Registration

A consumer must register with the server before he can subscribe for a data object. This registration is done only once for every consumer. The following steps are performed during registration.

1. The consumer C_i authenticates himself with the information server.
2. The server performs a table lookup to ensure that the consumer C_i is not already registered.
3. If the consumer is not already registered, the server generates a pair of encryption/decryption keys, (K_i, K_i^{-1}) , for the consumer C_i using algorithm 1 given below.
4. The server records the encryption key K_i in the table of registered users.
5. The server sends the decryption key K_i^{-1} to the consumer C_i . This decryption key is encrypted with the public key K_{pubi} and the encrypted key $([K_i^{-1}, K_{pubi}])$ is then transmitted.

Algorithm 1 Encryption, Decryption Key Generation for C_i

Input:

- (i) \mathbf{K} – List of encryption keys that have already been assigned to existing consumers,
- (ii) e – a large exponent chosen by the server which is common to all the consumers.

Output:

- (i) $\langle K_i, K_i^{-1} \rangle$ – Encryption Key K_i /Decryption Key K_i^{-1} for consumer C_i .

Procedure GenerateCustomerKeys(\mathbf{K}, e)

begin

```

    factors := 1
    for each  $K_j = \langle e, N_j \rangle$  in  $\mathbf{K}$  do
        factors := factors *  $N_j$ 
    choose an  $N_i$  that is relatively prime to factors
     $K_i := \langle e, N_i \rangle$ 
     $K_i^{-1} := \langle d_i, N_i \rangle$ 
    (where  $ed_i \equiv 1 \pmod{\phi(N_i)}$  as given in Def. 10)

```

end

Subscription

In this phase, the consumer actually subscribes to one or more data objects. Billing the consumer (which is beyond the scope of this paper) can also be done at this stage. The following steps are performed at the subscription phase.

1. For each data object the consumer C_i sends the server, the data object identification D_j and the period of subscription (T_{ijk}, T_{ijl}) .
2. The server maintains a list L_j for each data object D_j . Corresponding to each consumer C_k who has subscribed to the data item D_j there is a record R_{kj} in the list L_j . When a consumer C_i subscribes to a data object D_j for the first time, the server inserts a new record $R_{ij} = \langle C_i, D_j, K_i, (T_{ijk}, T_{ijl}) \rangle$. (Algorithm 2 describes how this is done). When a consumer C_i changes his subscription about the data object D_j , the existing record R_{ij} is modified (as shown in Algorithm 3).

Algorithm 2 Insert Consumer C_i 's Subscription

Input:

- (i) C_i – the identity of the consumer.
- (ii) \mathbf{D} – the set of data objects that the consumer is subscribing to.
- (iii) \mathbf{T} – the set of time intervals describing the period of subscription for each data object in \mathbf{D} .
- (iv) \mathbf{L} – A set of lists corresponding to the data objects in \mathbf{D} .

Output:

- (i) \mathbf{L} – The set of updated lists. Each list L_j will be updated by inserting the record R_{ij} .

Procedure InsertConsumerSubscription ($(C_i, \mathbf{D}, \mathbf{T}, \mathbf{L})$)

begin

```

for each data object  $D_j$  in  $\mathbf{D}$ 
    get  $\langle T_{ijp}, T_{ijq} \rangle$  from the set  $\mathbf{T}$ 
    if there is no record  $R_{ij}$  in list  $L_j$ 
        insert record  $R_{ij} = \langle C_i, K_i, (T_{ijp}, T_{ijq}) \rangle$  in  $L_j$ 

```

end

In our protocol the consumer has the flexibility of modifying or canceling his subscription at any time. This algorithm is given below.

Algorithm 3 Modify Consumer C_i 's Subscription

Input:

- (i) C_i – the identity of the consumer.
- (ii) \mathbf{D} – the set of data objects that the consumer is subscribing to.
- (iii) \mathbf{T} – the set of time intervals describing the period of subscription for each data object.
- (iv) \mathbf{L} – A set of lists corresponding to the data objects in \mathbf{D} .

Output:

(i) \mathbf{L} – The set of updated lists. Each list L_j will be updated by modifying the record R_{ij} .

Procedure ModifyConsumerSubscription $((C_i, \mathbf{D}, \mathbf{T}, \mathbf{L}))$

begin

for each data object D_j in \mathbf{D}

 get $\langle T_{ijp}, T_{ijq} \rangle$ from the set \mathbf{T}

if there is record R_{ij} in list L_j

 delete existing record R_{ij}

 insert record $R_{ij} = \langle C_i, K_i, (T_{ijp}, T_{ijq}) \rangle$ in L_j

end

Data Encryption and Broadcast

At each broadcast time t , the following activities are performed.

1. The server finds out the list of data objects that must be transmitted at this time.
2. For each of these objects, the server is responsible for generating the appropriate encryption key using the algorithm 4.
3. The data objects are encrypted with the correct key.
4. Each encrypted data object corresponds to a data block of the broadcast message.
5. The broadcast index block is created and prepended to the message. For each data item that is being broadcast, a pointer to the data block for that item is given in the broadcast index block. The structure of the broadcast message is given in figure 1.
6. The broadcast message is transmitted.

Algorithm 4 Broadcast Encryption Keys Generation

Input:

- (i) \mathbf{D} – the set of data objects,
- (ii) \mathbf{L} – the set of lists corresponding to all the objects.
- (iii) t – The broadcast time t .

Output:

- (i) The set of keys used to encrypt the data objects.

Procedure BroadcastEncryptionKeyGeneration $(\mathbf{D}, \mathbf{L}, t)$

begin

for each data object $D_j \in \mathbf{D}$ **do**

$factor_j := 1$

for each list L_j in \mathbf{L}

for each record R_{ij} in L_j **do**

if t is within the interval (T_{ijk}, T_{ijl}) **then**

$factor_j = factor_j * N_i$

(where N_i is the modulus of Key K_i)

if $(factor_j \neq 1)$ **then**

$Key_{D_j} := \langle e, factor_j \rangle$

else no key to generate for data D_j .

end

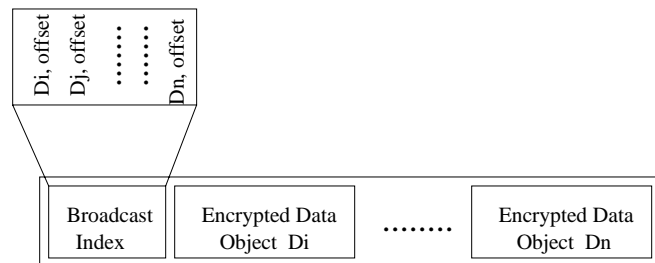


Figure 1. Structure of the Broadcast Message

Receipt of the Data Objects

In order to retrieve the data object during the subscription interval, the consumer does the following.

1. The consumer C_i tunes in and listens to the broadcast message.
2. The consumer reads the broadcast index to find out which data block(s) contain the data object(s), the consumer is interested in.
3. The consumer then gets the encrypted data object(s) from the respective data block(s).
4. To decrypt each data object, the consumer C_i uses his decryption key K_i^{-1} . (Theorems 2 and 3 ensure that this is possible).

4.1. Optimizations

The server may choose to do certain optimizations to improve the performance. For example, each data object D_j has a list L_j that records the details of the consumers who have subscribed to this object. Rather than maintain a single list, the server can partition the list based on subscription times. During each broadcast time, the appropriate partition will be traversed.

The server can also perform certain optimizations to reduce the size of the broadcast. The size of the broadcast message depends on the number of data objects that are being transmitted (it is independent of the number of subscribers subscribing to that data object in that time interval). Often, it may happen that two or more data objects are being

encrypted with the same key (this may happen, if the same set of consumers have subscribed to the same data objects). In such a case, these data objects can be concatenated and their result encrypted – this way the consumer does not have to perform multiple decryptions for the set of objects.

4.2. Security of Our Scheme

Our scheme is based on the RSA cryptosystem. Its security is based on the difficulty of factoring large prime numbers. We do need to mention that the low exponent attack on the RSA cryptosystem [12] does not arise in our case. The low exponent attack occurs in the following way: suppose the same message m is encrypted with different keys sharing the same exponent. Let the exponent $e = 3$ and the different keys are $K_1 = \langle e, N_1 \rangle$, $K_2 = \langle e, N_2 \rangle$, $K_3 = \langle e, N_3 \rangle$, etc. By using the Chinese Remainder Theorem [13] an attacker can get m^e . Now if he can guess e correctly, then by extracting the e th root of m^e , he can obtain m .

Note that in our mechanism, the same data is not encrypted multiple times with different keys and retransmitted. In fact, the data is encrypted once using one key. Thus, having multiple copies of the same data encrypted with different keys does not arise in our case. We also choose a very large exponent as an additional precaution.

5. Conclusion and Future Work

In this work we proposed a new protocol for secure multicasting. Our protocol is scalable - the length of the broadcast message is independent of the number of consumers subscribing to a data object. Key management is simple: each consumer has to remember one key only; the number of keys the server has to store equals the number of consumers. Our protocol does not require key distribution for every session or during group changes. If the group of consumers subscribing to an object changes, the existing group members are not aware of this change.

A lot of work remains to be done. First, we plan to formally specify and analyze the protocol. This may reveal some undetected flaws that need to be resolved. Next we plan to do a simulation that will allow us to compare the performance of this protocol with existing ones. Finally, we plan to implement the protocol. Implementation will reveal the actual shortcomings of the protocol, if any.

References

- [1] M. V. D. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System . In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer-Verlag, 1995.
- [2] J. L. Camenisch. Efficient and Generalized Group Signatures. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer-Verlag, 1997.
- [3] J. L. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups. In *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.
- [4] A. Celik and A. Datta. A Scalable Approach for Subscription-Based Information Commerce . In *Proceedings of the 2nd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, Milpitas, CA, June 2000.
- [5] D. Chaum and E. V. Heyst. Group Signatures . In D. W. Davies, editor, *Advances in Cryptology – EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.
- [6] L. Chen and T. P. Pederson. Group Signatures. In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 171–181. Springer-Verlag, 1995.
- [7] G. Chiou and W. Chen. Secure Broadcasting Using The Secure Lock. *IEEE Transactions on Software Engineering*, 15(8), August 1989.
- [8] A. Fiat and M. Naor. Broadcast Encryption . In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer-Verlag, 1994.
- [9] I. S. Gopal and J. M. Jaffe. Point-to-Multipoint Communication Over Broadcast Links. *IEEE Transactions on Communications*, 32(9):1034–1044, 1984.
- [10] I. Ingemarsson, D. T. Tang, and C. K. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5):714–720, September 1982.
- [11] M. Just and S. Vaudenay. Authenticated Multi-Party Key Agreement. In *Advances in Cryptology – ASIACRYPT '96*, *Lecture Notes in Computer Science*, pages 36–49. Springer-Verlag, 1996.
- [12] B. Kaliski and M. Robshaw. The Secure Use of RSA. *CryptoBytes*, 1(3):7–13, 1995.
- [13] I. Niven and H. S. Zuckerman. *An Introduction to the Theory of Numbers*. John Wiley and Sons, 4th edition, 1980.

- [14] W. Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice-Hall, 2nd edition, 1999.
- [15] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 31–37, New Delhi, India, March 1996.