Dynamic Elgamal Public Key Generation with Tight Binding

by R. Poovendran, M. Scott Corson and John S. Baras

CSHCN T.R. 99-38
(ISR T.R. 99-79)

Proceedings

Third Annual Federated Laboratory Symposium on Advanced Telecommunications/ Information Distribution Research Program (ATIRP)

February 2-4, 1999
College Park, MD

DYNAMIC ELGAMAL PUBLIC KEY GENERATION WITH TIGHT BINDING*

R. Poovendran, M. S. Corson, J. S. Baras

Institute for Systems Research,
University of Maryland at College Park
College Park, MD 20742
e-mail: {radha, corson, baras}@isr.umd.edu

ABSTRACT

We present a new distributed, group ElGamal public key generation algorithm which also permits maintenance of a group-specific, dynamic, individual ElGamal public key infrastructure. We parameterize the group with a time-varying quantity that serves as a distributed mechanism for controlling key generation privilege. Our scheme can be viewed as an alternative to polynomial schemes where, at the time of the secret construction step, there has to be a third party or a black box to combine the shares. Also, in polynomial schemes, at the time of combining, the individual shares of the secret have to be revealed to the third party. In our scheme, the common secret can be generated without ever exposing the individual shares constructing it. We note that many of the recently proposed distributed key management [2-4] schemes need such group keys for certification and signing purposes.

INTRODUCTION

In this paper, we present a key generation algorithm that allows mutually-suspicious members to generate individual ElGamal public keys, and to jointly generate a group ElGamal public key. This algorithm is based on a non polynomial concept, first presented in [1], that demonstrates the feasibility of allowing mutually-suspicious parties to generate a common key. The keys are generated in a dynamic (i.e. time-varying) manner, and every member contributes a share (i.e. a secret) in generating the group key. The resulting key system is a dynamic, bi-level (individual and group), group-contained public key infrastructure. We show how to protect the shares of the individual members involved in generating the keys. We also show how the generation of the public keys of individual members can be tied to the group key generation procedure. In terms of security features, the new scheme has at least the same level of guarantees as the basic ElGamal approach and—because of the nature in which we generate the keys—embeds *non-cryptographic* security features into it, further enhancing security. For example, breaking the group key for all key updates in our method is at least equivalent to breaking n different ElGamal keys. Breaking the group public key alone does not allow the attacker to get access to any future keys.

BASIC IDEA BEHIND OUR SCHEME

Our scheme enables a set of n specific, mutually suspicious, members to generate a common shared secret such that each member contributes equally to the common secret without ever having to reveal it to any other member. The only way to extract the individual share of a member without his/her consent is for all the remaining members to collude and also use the information contained in the current common secret.

Our scheme consists of the following ideas:

1. Providing individual members with a dynamic *padding* that helps to hide the individual shares.
2. Making sure that the padding parameters for every member are different.
3. Making sure that the padding parameter is not derived from any functions, such as one-way hashes.
4. Each member can locally generate its pads at each iteration.
5. All the participating members can, after combining the *hidden* or *padded* shares of other members, use a group binding parameter that is dynamic, and locally computed to remove the net effect of the pads without ever being able to know the individual shares of other members.

In order to use such a padding mechanism, we need to make sure that the padding will lead to *perfect secrecy* or *Shannon secrecy*, as long as the pads and the individual shares are chosen independently and uniformly over the valid parameter range with appropriate *modulo arithmetic operation*. One important feature of this condition is that the bit length of the pads should be same as the bit length of the individual secrets. Once this condition is satisfied, adding a pad to the secret will lead to another uniformly

*Prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federal Research Laboratory Program, Cooperative Agreement DAAL01-96-0002"

distributed random number of same bit length which will also be independent of the pads and the individual shares. Hence, other members who receive the hidden shares will not be able to decompose it to obtain the pad or the *true* share of a member.

The condition that all the members be able to generate the common shares locally and still remove the effect of the pad may appear non-feasible at first glance since we are not defining the pads to be derived from any standard cryptographic functions such as hashes. We note that use of any such function will lead to problems since, (1) if the padding function is known to any other member, in order to remove the padding effect at any time the member also needs to know the input parameters to the hash function used and hence, nothing can be hidden from any member, (2) if an external attacker collects enough ciphers and succeeds in breaking the secret and obtains the inputs to the hash function at one time, he/she will know it for all other times.

Hence, the padding function should not have any structure. However, every valid member must be able to remove the cumulative effect of the padding and use only the shares to generate the new common secret. In order to achieve this goal, each member should be able to locally compute an appropriate *group parameter* that will, when suitably applied, nullify the effect of the cumulative padding effect. This parameter should be such that it is, dynamically updated at each common secret update step, locally computed by each member and not transmitted except at the time of initialization, able to remove only the cumulative effects of all the pads. Furthermore, to prevent any participating member from stealing the secrets of other members, the group parameter should never be able to remove the padding effect when applied to the combined share that is generated by combining contributions from less than n members.

This paper shows how these goal are actually achievable. Apart from the mathematics presented, the necessary algorithm can be summarized for iteration j for a member index i as follows:

- generate new individual shares
- combine it with the individual dynamic pad to generate hidden shares
- exchange hidden shares of all the members securely
- compute the new common secret with the pads by combining all the hidden shares
- use the group binding parameter to remove the cumulative effect of padding.
- use the new common secret to locally update the padding and the group parameter.

PROPERTIES OF THE NEW KEY GENERATION SCHEME

The following notation is used to describe the different quantities used in the algorithm:

$\alpha_{i,j}$: The one-time pad of the ith member at the jth key update iteration.

θ_j: The pad binding parameter at the jth key update iteration.

$\{K_i, K_i^{-1}\}$: Public key pair of the member i. This pair is assumed to be updated appropriately to key the integrity and confidentiality of any communication transaction by and with member i.

$FK_{i,j}$: The Fractional Key (FK) of the ith member at the jth key update iteration.

$HFK_{i,j}$: The ***hidden*** FK (HFK) of the ith member at the jth key update iteration.

SK_j: The group Shared Key (SK) at the jth key update instance.

$A \longrightarrow B : \mathcal{X}$: Principal A sends principal B a message $\mathcal{X}$.

Our message format is $\{\{T_i, M, j, Msg\}_{K_S^{-1}}\}_{K_R}$, where

- T_i: a real-valued, wallclock time stamp generated by member i.
- M: denotes the *mode* of operation with "I" for Initialization mode, "G" for key Generation mode, and "R" for key Recovery mode.
- j: integer-valued, denotes the current iteration number.
- Msg: the message to be sent.
- K_S^{-1}: Denotes the private key of the sender S.
- K_R: Public key of the receiver.

In developing the new key scheme, we note that the following properties are desirable for a multiparty key generation scheme:

- A FK contributed by a participating member should have the same level of security as the group SK.
- A single participating member, without valid permissions, should not be able to obtain the FK of another member.
- If a FK-generating member has physically failed, been compromised or removed, the remaining FK-generating members should be able to jointly recover the FK of the failed member (this requires not majority voting but total participation).

We note that the first property simply states that the distributed key generation scheme has to be such that each FK space has at least the same size as the final secret key space. Hence, each member may generate FK of different size but, when combined, they lead to a fixed length shared secret.

The second property has to do with the need for protection of individual FKs that is desired due to the absence of a centralized key generation scheme. In the current scheme, every member performs an operation to *hide* its FK such that, when all the *hidden FKs* (HFK) and the group parameter are combined, the net result is a new secret key. We note that even if a HFK is known, the problem of obtaining the actual FK or the secret key needs further computation. We will describe the requirements of the FK concealment mechanism in the next section.

If a contributing member physically fails, becomes compromised, or has to leave the multicast group, then it becomes necessary to replace the existing member with a new member. Hence, the newly-elected member should be able to securely recover the FK generated by the replaced member. We note that the requirement that an individual member acting alone not be able to obtain the FKs of other contributing members is similar to protecting individual private keys in the public key crypto systems.

DESCRIPTION OF THE MULTIPARTY KEY GENERATION SCHEME

The following is a list of assumptions regarding the algorithm, some of which may appear rather abstract at first glance:

- There exist a commutative operator $\circledcirc$ which forms a commutative group, also commonly known as *Abelian group*, when operating on the set of keys.
- It is computationally difficult to perform crypt-analysis on a cryptographically-secure random key by search methods if the key length is sufficiently large.
- The keys are all L bits in length, and all members know this length.
- The number of participants in generating the secret key is fixed as n (where n may be a function of $\circledcirc$).
- There is a mechanism for certifying the members participating in the key generation procedure, for securely exchanging the quantities required in the algorithm and for authenticating the source of these quantities.
- Every member has the capability to generate a cryptographically-secure random number, or a fresh quantity, of length at least L bits.[†]

With the assumptions above, we note that the key management scheme consists of three major parts:

1. Initialization—consisting of member selection, and secure initial pad and binding parameter generation and distribution;
2. Key Generation—an iterative process consisting of fractional, hidden and shared-key generation; and
3. Key Retrieval—required only in the case of a member node failure or compromise.

INITIALIZATION ALGORITHM

A Group Initiator (GI) first selects a set of n FK-generating members, and the GI may be one of these members (how it occurs is not specified and is application-dependent). The GI then either (1) contacts a Security Manager (SM)—a third party who is *not* a FK-generating member—who generates the initial pads and the binding parameter and distributes them to the members, or (2) initiates a distributed procedure among the group members to create these quantities without the aid of a third party. We will focus on only the SM based initialization.

THIRD PARTY-BASED INITIALIZATION

The initial pads and binding parameter are distributed to each member i, for $i = 1, \ldots, n$, as

$$SM \longrightarrow i : \{\{T_{SM}, I, 1, \alpha_{i,1}, \theta_1\}_{K_{SM}^{-1}}\}_{K_i}$$

where $\alpha_{i,1}$—its initial one-time pad—is computed such that $\alpha_{1,1} \circledcirc \alpha_{2,1} \circledcirc \cdots \circledcirc \alpha_{n,1} = \theta_1$.

[†] By cryptographically secure, we mean that the generated keys are not in set of weak keys for the intended encryption/decryption algorithm. For example, if the generated key is such that if all bits are ones or all are zeros, then that key may be the easiest first guess of an attacker. Main idea is to make sure that the key has the highest possible entropy and hence the highest possible amount of randomness in its bits.

KEY GENERATION ALGORITHM

The key generation algorithm is an iterative process depicted in Figure 1. Each iteration j requires as input (indicated as step (0) in the figure) a set of one-time pads $\alpha_{i,j}$, $i = 1, \ldots, n$, and the binding parameter θ_j, which are obtained from the initialization algorithm for iteration $j = 1$, and from the preceding iterations for $j > 1$. For simplicity we choose $\circledcirc = \mathit{mod}\, p$, where p is a large odd prime with $p - 1$ having very large prime factors.

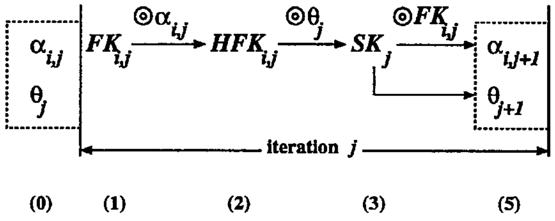

Figure 1. Iteration and mappings of the key generation algorithm

The iterative key generation algorithm consists of the following steps (1)-(5):

1. For $i = 1, \ldots, n$, a member i randomly picks up a number $FK_{i,j}^{-1}$ with $0 \leq FK_{i,j}^{-1} \leq p - 2$ and generates $FK_{i,j} = \alpha^{FK_{i,j}^{-1}}$. Here, $(FK_{i,j}^{-1}, FK_{i,j})$ is an individual ElGamal public key pair for the member i at time update j.
2. For $i = 1, \ldots, n$, a member i generates a quantity $HFK_{i,j} = (\alpha_{i,j} + FK_{i,j}) \bmod p$, and then all members securely exchange the HFKs and their public FK as $\forall\ 1 \leq l, m \leq n$, $l \neq m$,
   $l \longrightarrow m$: $\{\{T_l, G, j, HFK_{l,j}, FK_{l,j}\}_{FK_{l,j-1}^{-1}}\}_{FK_{m,j-1}}$.
3. Once the exchange is complete, each member computes the θ_{j+1} as
   $\theta_{j+1} = ((p - n - 3)\theta_j + \sum_{i=1}^{i=n} HFK_{i,j}) \bmod (p-1)$.
   $\Rightarrow GK_{j+1}^{-1} = \theta_{j+1}$
   $GK_{j+1} = \alpha^{GK_{j+1}^{-1}} = \prod_{i=1}^{i=n} FK_{i,j} = \prod_{i=1}^{i=n} \alpha^{FK_{i,j}^{-1}}$.
4. If the resulting group key pair is cryptographically-insecure for a particular application, all members can repeat steps (1) - (3) creating a new high quality key pair.
5. For $i = 1, \ldots, n$, a member i computes the iteration update as
   $\alpha_{i,j+1} = (GK_j^{-1} + FK_{i,j}^{-1}) \bmod p$.

The steps (1) - (5) present the computational steps for generating the keys at each update. At the end of step (5), we have the group private key for the current iteration. We note the following additional features of the key scheme:

- Although all the members have each $HFK_{i,j}$, obtaining the $FK_{i,j}$ or $\alpha_{i,j+1}$ of another member involves search in the L-dimensional space, and obtaining their correct combination involves $(n-1)$ searches in L-dimensional space. Hence, even if a fellow member becomes an attacker, that rogue member faces nearly the same computational burden in obtaining the set of n FKs as an outside crypto analyst; i.e. trust is *not* unconditional.
- For such an outside attacker, breaking the system requires either a search in a L-dimensional space to get θ, or in n L-dimensional searches to break individual secrets of all the members. Access to all n HFKs is alone insufficient to permit an attacker to determine the secret key; for that, the attacker must also possess the current binding parameter θ which is time-varying and never transmitted.

CHOICE ON THE NUMBER OF MEMBERS

Regarding the choice of the number of members, clearly, the choice of $n = 2$ is not appropriate for such a scheme. Although choosing $n = 3$ does not instantly expose a secret pad α; when a participating member becomes an attacker (i.e. a *rogue*), the following attack—called *fractional attack* (FA)—is feasible.

Lemma: When $\circledcirc$ is an $\oplus$ function, independent of how non-trivial the bit-length of the key is, choosing $n = 3$ permits a FA.

Proof: Assume that the time instant at which one member i ($i = 1$ or 2 or 3) becomes a *rogue* is j. At this time the members have values of $\alpha_{1,j} = HFK_{2,j} \oplus HFK_{3,j}$, $\alpha_{2,j} = HFK_{3,j} \oplus HFK_{1,j}$, $\alpha_{3,j} = HFK_{1,j} \oplus HFK_{2,j}$. Every member also has access to the current θ_{j+1} and their own $FK_{l,j}$ ($l = 1, 2, 3$). At this stage, obtaining the α component of any other member is as computationally intensive as an outside attacker trying to obtain θ_{j+1}. However, if a member, say $i = 1$, is compromised and releases its secret $\alpha_{1,j}$, then each of the other members can use this and compute $FK_{1,j} = \alpha_{1,j} \oplus \theta_j$. Since the $\theta_{j+1} = FK_{1,j} \oplus FK_{2,j} \oplus FK_{3,j}$, each member can now compute the other non-rogue member's FK as well.

This leads to the following
Corollary: When $\circledcirc$ is an $\oplus$ function, independent of how non-trivial the bit-length of the key, the minimum number of members to prevent a FA by a single *rogue* member for the multiparty key scheme is 4.

VERIFIABLE SECRET SHARING FOR KEY GENERATION SCHEME

Since there are multiple entities involved in key generation, it becomes important to have a mechanism to verify if the parameters exchanged do contribute to the generated shared key. The verification steps have to be followed at (1) SM-based group initialization, (2) Distributed Group initialization, (3) θ-generation iteration and (4) key recovery.

SM-based Initialization

In the case of the SM-based scheme, each member i needs to make sure that the SM uses non-trivial values for its $\alpha_{i,1}$ and θ_1. Since each member needs to protect its individual pad value, one method for openly checking correctness of the pads is to generate a public value that will enable all the key generating members to check their correctness without revealing the actual value of the individual pads. Such a verification technique falls under the category of Verifiable Secret Sharing (VSS) [5, 6].

If one wants to check if the individual initial pads $\alpha_{i,1}$ given by the security manager are "good", the scheme given below can be used.

1. SM member generates $(\widehat{\alpha}_{i,1}, \widehat{\theta_1}) = (g^{\alpha_{i,1}}, g^{\theta_1})$ and sends the result to all the members (g is the generator of the group and we assume that it is difficult to perform the discrete logarithm within the time of interest).
2. Each member i can privately verify that $g^{\theta_1} = g^{\widehat{\theta}_1} = \prod_{j=1}^{j=n} \widehat{\alpha}_{i,1} = g^{\theta_1}$ and $\widehat{\alpha_{i,1}} = g^{\alpha_{i,1}}$, where failure (inequality) means that some or all of the given pads don't correspond to the given θ_1.

CONCLUSIONS AND FUTURE WORK[‡]

We have presented a distributed, group ElGamal public key generation algorithm which also permits maintenance of a group-specific, dynamic, individual ElGamal public key infrastructure, by using the concept of *fractional keys*, first introduced in [1]. We parameterize the group with a time-varying quantity that serves as a distributed mechanism for controlling key generation privilege. We also noted that the new scheme forces the attacker to break n keys to have access to the future group keys update process. Our scheme can be viewed as an alternative to polynomial schemes where at the last step there has to be a third party or a black box to combine the secret. In polynomial schemes, at the time of combining, the individual shares of the secret have to be revealed to the third party. In our scheme the secret can be extracted without revealing the individual shares constructing it. Fundamentally, in the polynomial scheme, if a third party has access to all the shares, the third party can uniquely reconstruct the secret. Such is not the case in our scheme; the third party not only needs the individual shares but also the group binding parameter.

[‡] "The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

REFERENCES

[1] R. Poovendran and S. Corson and J. Baras, "A Distributed Shared Key Generation Procedure Using Fractional Keys", Proceedings of the MILCOM'98, OCT, 1998, Boston MA.

[2] H. Harney, C. Muckenhirn, "GKMP Architecture", RFC 2093, July 1997.

[3] H. Harney, C. Muckenhirn, "GKMP Specification", RFC 2094, July 1997.

[4] A. Ballardie, "Scalable Multicast Key Distribution", RFC 1949, May 1996.

[5] P. Feldman, "A Practical Scheme for Non-Interactive Verifiable Secret Sharing", *Proc. of IEEE Fund. Comp. Sci.*, 427-437, 1987.

[6] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing", *Advances in Cryptology* - CRYPTO, LNCS 576:129-140, 1991.