A DISTRIBUTED SHARED KEY GENERATION PROCEDURE USING FRACTIONAL KEYS

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

Center for Satellite and Hybrid Communication Networks
Institute for System Research,
University of Maryland at College Park
College Park, MD 20742
e-mail: {radha, corson, baras}@isr.umd.edu

ABSTRACT

We present a new class of distributed key generation and recovery algorithms suitable for group communication systems where the group membership is either static or slowly time-varying, and must be tightly controlled. The proposed key generation approach allows entities which may have only partial trust in each other to jointly generate a shared key without the aid of an external third party. The shared key is generated using strong one-way function of the group parameter. This scheme also has perfect forward secrecy. The validity of key generation can be checked using verifiable secret sharing techniques. The key retrieval method does not require the keys to be stored in an external retrieval center. We note that many Internet-based applications may have these requirements. Fulfillment of these requirements is realized through the use of fractional keys—a distributed technique recently developed to enhance the security of distributed systems in a non-cryptographic manner.

INTRODUCTION

Cryptographic key generation and management is an important problem in multicast and group communications [1-5]. In many instances, it is desirable to generate a group *shared key* (SK) for efficient intra-group communications. However, having the same SK implies that the all the group membership is at the same trust level. In a distributed, multicast group, it is often not possible nor desirable to have the *same* trust level throughout the group. One may be tempted to suggest that a single trust level can be defined by choosing the lowest possible trust level as the group trust level. Though such a straightforward approach is feasible, one can do better by compartmentalizing the group based on local trust levels [5]. Such a compartmentalization inevitably leads to *clustering* of a given group. Compartmentalization also helps in having a better control over the set of key management and distribution functionalities as noted in [5].

While the entities in each cluster may share a common trust level, it may be that the clusters are mutually suspicious and have only *partial* trust in each other. Thus, a mechanism is desired that permits mutually suspicious parties to come together to generate a shared key. In order to avoid involving (and paying) a third party, it is also desirable that the scheme involve only the group members and no external parties.

Schemes in [2,3,4] propose to replace the traditional (external) Key Distribution Center (KDC) with a *Group Controller* (GC) which can generate and distribute the keys. However, in these approaches, a single member is allowed to generate the keys. This means that group members must place complete trust in this group member. In [5], a *panel* of members are allowed to generate the keys. However, [5] does not present any explicit distributed key generation scheme.

In this paper, we present a class of key management schemes which increase the security of key generation and recovery using non-cryptographic techniques. The schemes employ distributed algorithms based on *Fractional Keys* (FK). The proposed methods allow the members to automatically update the keys in a periodic manner without any assistance from an external third party, and use verifiable secret sharing techniques in [7,8].

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 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 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 nonce 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, 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 SK space. Hence, each member may generate FK of different size but, when combined, they lead to a fixed length SK.

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 perform 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 SK. We note that even if a HFK is known, the problem of obtaining the actual FK or the SK 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.

However, to ensure the integrity of the scheme, this recovery should be possible only if all the remaining contributing members cooperate. This feature deviates significantly from the existing key generating schemes [2,3,4]. 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 two commutative operators ⊛ and ◇ which form an Abelian group when operating on the key elements.

- It is computationally difficult to perform crypto 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 SK is fixed as n (where n may be a function of ⊛ and ◇).

- 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 Recovery—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.

SECURITY MANAGER-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 $\alpha_{1,1} \otimes \alpha_{2,1} \otimes \cdots \otimes \alpha_{n,1} = \theta_1$.
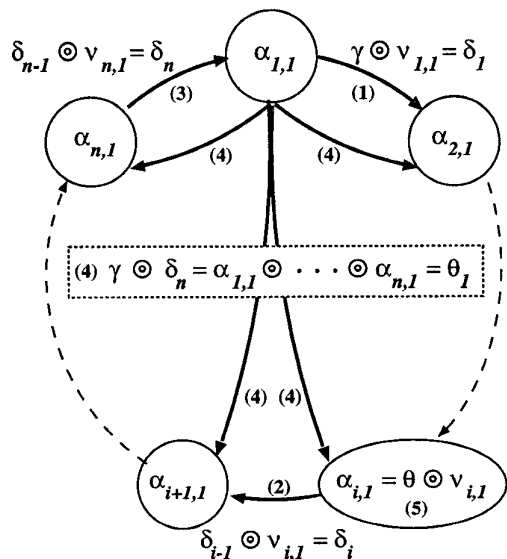
DISTRIBUTED INITIALIZATION

Figure 1. Distributed initialization algorithm

The GI (assumed to be a member and denoted here by the index 1 shown in Figure 1) can perform the following steps (1)-(5) to generate the initial parameters of the group:

1. Generate two uniformly-distributed random quantities γ and $\nu_{1,1}$ of bit length L, operate on these two quantities as $\gamma \otimes \nu_{1,1} = \delta_1$, and send the result to member 2 (the "next" member in the group) as $1 \longrightarrow 2$: $\{\{T_1, I, 1, \delta_1\}_{K_1^{-1}}\}_{K_2}$.

2. The following steps are repeated for $i = 2, \ldots, n - 1$:

   (a) Member i generates a uniform random variable $\nu_{i,1}$ of bit length L.

   (b) Member i then operates on the quantity it received from member $i - 1$ as $\delta_{i-1} \otimes \nu_{i,1} = \delta_i$.

   (c) Member i then sends the result to member $i + 1$ as $i \longrightarrow i + 1$: $\{\{T_i, I, 1, \delta_i\}_{K_i^{-1}}\}_{K_{i+1}}$.

3. Eventually, the group member $i = n$ receives δ_{n-1} and then generates a uniformly-distributed random quantity $\nu_{n,1}$ of bit length L, performs $\delta_{n-1} \otimes \nu_{n,1} = \delta_n$, and then securely sends it to the initiating member $i = 1$ as $n \longrightarrow 1$: $\{\{T_n, I, 1, \delta_n\}_{K_n^{-1}}\}_{K_1}$.

4. The initiator (member 1) then decrypts it and performs $\gamma \otimes \delta_n = \theta_1$, and then sends θ_1 to each member i, for $i = 2, \ldots n$, as
   $1 \longrightarrow i : \{\{T_1, I, 1, \theta_1\}_{K_1^{-1}}\}_{K_i}$.

5. Each member i privately computes $\alpha_{i,1} = \theta_1 \otimes \nu_{i,1}$, and uses $\alpha_{i,1}$ as its initial pad.

We note that these two approaches of initialization—security manager-controlled and distributed—are not equivalent unless additional security assumptions are made. For example, in the case of distributed initialization within the group, we point out that using following attack is feasible.

Assume that members $i - 1$ and $i + 1$ conspire to obtain the secret of member i, where the numerical ordering corresponds to the order of message passing in the distributed algorithm.

1. Member $i - 1$ sends δ_{i-1} to member i as per the algorithm, and *also* to member $i+1$ without i's knowledge.

2. Member i, who is unaware of the conspiracy between $i - 1$ and $i + 1$, computes $\delta_i = \delta_{i-1} \otimes \nu_{i,1}$ and sends it to member $i + 1$ securely.

3. Member $i + 1$ can now compute $\nu_{i,1} = \delta_{i-1} \otimes \delta_i$ and obtain the secret $\nu_{i,1}$ of member i.

However, the secret $\nu_{i,1}$ generated by member i becomes part of the pads (i.e. the α's) of members $i - 1$ and $i + 1$. Hence, the knowledge of $\nu_{i,1}$ reduces the entropy of the initial pads of the conspiring members. Thus, while the attack is feasible, there may not be any incentive to conspire in this manner.

KEY GENERATION ALGORITHM

The key generation algorithm is an iterative process depicted in Figure 2. 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
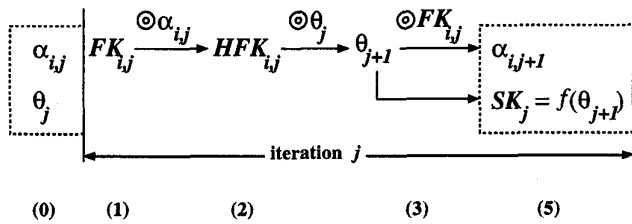
Figure 2. Iteration and mappings of the key generation algorithm

are obtained from the initialization algorithm for iteration $j = 1$, and from the preceding iterations for $j > 1$.

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

1. For $i = 1, \ldots, n$, a member i generates a cryptographically-secure random number $FK_{i,j}$.

2. For $i = 1, \ldots, n$, a member i generates a quantity $HFK_{i,j} = \alpha_{i,j} \circledcirc FK_{i,j}$, and all the members securely exchange the HFKs as
$\forall 1 \leq l, m \leq n, l \neq m,$
$l \longrightarrow m: \{\{T_l, G, j, HFK_{l,j}\}_{K_l^{-1}}\}_{K_m}.$

3. Once the exchange is complete, each member computes the new group parameter θ_{j+1} as
$\theta_{j+1} = \theta_j \circledcirc HFK_{1,j} \circledcirc HFK_{2,j} \circledcirc \cdots \circledcirc HFK_{n,j}.$
$\Rightarrow \theta_{j+1} = FK_{1,j} \circledcirc FK_{2,j} \circledcirc \cdots FK_{n,j}.$

4. If the resulting group parameter θ_{j+1} is cryptographically insecure for a particular application, all members can repeat steps (1) - (3) creating a new high quality group parameter θ_{j+1}.

5. For $i = 1, \ldots, n$, a member i computes $\alpha_{i,j+1} = \theta_{j+1} \circledcirc FK_{i,j}$, and $SK_j = f(\theta_{j+1})$ where $f(.)$ is a strong one-way function.

The steps (1) - (5) present the computational steps for generating the keys at each update. At the end of step (5), we have the SK for the current iteration. Note that the quantity $\alpha_{i,j+1}$ is computed such that, for an outsider, obtaining $\alpha_{i,j+1}$ is much harder even if the actual key SK_j is compromised at any key update time interval $(j, j + 1)$. Knowing the group key does not reveal the group parameters and hence the tight binding of the members will not be broken by the loss of the shared key!. 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 $\alpha_{i,j+1}$ involves search in the L-dimensional space. Hence, even if a fellow member becomes an attacker, that rogue member has the same

amount of computational burden in obtaining the FK as a crypto analyst; i.e. trust is not unconditional. WE note that the burden of breaking all the (n-1) terms simultaneously leads to a (n-1)L dimensional search although the key is in L dimensional space.

- Even if an outsider captures and decrypts a packet and obtains the HFK of a single participating member, (a) having a HFK does not give any advantage to the crypto analyst in decrypting any message encrypted with the SK, (b) attacker has to find the corresponding remaining time varying $n - 1$ HFKs. Such is the case since the keys are transported in a secure manner. Hence, only the participating members have the direct access—maybe after decrypting—to the HFKs. For an outsider, it may be much harder to simultaneously attack and obtain these $n - 1$ parts since the search space will be $(n - 1)L$ dimensional.

RETRIEVAL OF THE FRACTIONAL KEY AND PAD OF A FAILED NODE

The following steps are involved in recovery of the $FK_{\tilde{i},j}$ and $\alpha_{\tilde{i},j}$ of the node failed $\tilde{i}$, where j represents the iteration number in which the node was compromised or failed.

1. Any one FK-generating member—called the Recovery Initiator (RI)—must initiate recovery and give the HFK of the failed node $\tilde{i}$ to the newly-elected node i as $RI \longrightarrow i : \{\{T_{RI}, R, j, HFK_{\tilde{i},j}\}_{K_{RI}^{-1}}\}_{K_i}.$

2. The RI must also give the newly-elected node i the current SK as $RI \longrightarrow i : \{\{T_{RI}, R, j, SK_j\}_{K_{RI}^{-1}}\}_{K_i}.$

3. Using the same algorithm as is used for distributed initialization, with the following replacements: (a) θ by ξ and (b) $\alpha_{l,j}$ by $\beta_{l,j}$. Except for the changes in the notation and the number of members participating, the algorithm for pad generation is same as for distributed initialization. Hence, at the end of this distributed pad generation, each member l has $\beta_{l,j}$ as its pad for key recovery process, and all these pads are bound with the parameter ξ.

4. For $l = 1, \ldots, n - 1$, each node l then computes a *modified* hidden fractional key $\widehat{HFK}_{l,j} = \beta_{l,j} \diamond FK_{l,j}$ and hands it to the newly-elected member i as $l \longrightarrow i : \{\{T_l, R, j, \widehat{HFK}_{l,j}\}_{K_l^{-1}}\}_{K_i}.$

5. Node i then combines all of the modified HFKs and recovers the fractional key $FK_{\tilde{i},j}$ using the operation
$FK_{\tilde{i},j} = \xi \diamond \widehat{HFK}_{1,j} \circledcirc \cdots \circledcirc \widehat{HFK}_{n-1,j} \circledcirc \theta_{j+1}.$

6. Node i then extracts the pad $\alpha_{\tilde{i},j}$ using the operation
$\alpha_{\tilde{i},j} = HFK_{\tilde{i},j} \circledcirc FK_{\tilde{i},j}.$

We note that the recovered values of $FK_{\bar{i},j}$ and $\alpha_{\bar{i},j}$ are unique. Once the new node recovers the fractional key of the compromised node, it can inform the other contributing members to update the iteration number j to $j+1$, and then all members can execute the key generation algorithm. Note that even though the newly-elected member recovers the compromised fractional key and pad, the next key generation operation of the new node does not use the compromised key or pad. Hence, even if the attacker possesses the fractional key or pad at iteration j, it does not allow the attacker to obtain the future fractional keys or pads without any computation.

A SPECIFIC CHOICE OF THE FUNCTIONS ⊚ AND ◇

We have presented a class of multiparty key generation algorithms where a given instance of the class is determined by choice of functions ⊚ and ◇. Depending on the choice of these functions, the lower bound on n may be determined.

We note that among the possible choices for both ⊚ and ◇ are the modulo 2 addition (or XOR) function when n is even, and the modulo 3 addition function when n is odd— both of which we denote here with $\oplus$ simply meaning modulo addition.

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 ⊚ 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 ⊚ 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 actually contribute to the generated shared key. The verification steps have to be followed at (1) SM-based group initialization, (b) Distributed Group initialization, (c) SK-generation iteration and (d) 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 VSS [7, 8].

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. Any one member (possibly the SM) picks a very large prime number p and sends it to all the members. The number picked should larger than the possible range of the SK value. The same member also sends a generator g of the multiplicative group under p.

2. Each member i picks a random polynomial f_i with value 0 at the origin.

3. Each member i adds the polynomial value to the pad value, generates $\widehat{\alpha}_{i,1} = g^{\alpha_{i,1} + f_i}$ and sends the result to all the other members.

4. Each member i computes $g^{\widehat{\theta}_1} = \prod_{j=1}^{j=n} \widehat{\alpha}_{i,1} = g^{\theta_1 + \sum_{j=1}^{j=n} f_j}$ and evaluates it at origin to check if the value is equal to g^{θ_1}.

5. Each member i checks if $g^{\alpha_{i,1}} \stackrel{?}{=} \frac{g^{\theta_1}}{\prod_{j=1}^{n} g^{\alpha_{j,1}}}$, where failure (inequality) means that some or all of the given pads don't correspond to the given θ_1.

We note that it is also possible to use specific polynomial based techniques to allow members to verify if the individual pads are correctly distributed to the members.

Distributed Initialization

In the case of distributed initialization, the following scheme can be used to check if the GI gives the θ that is generated from the contributions of the group members.

1. Any one member (possibly the GI) picks a very large prime number p and sends it to all the members. The number picked should be larger than the possible range of the SK value. The same member also sends a generator g of the multiplicative group under p.

2. The GI computes g^γ and $g^{\nu_{1,1}}$, and makes it available to all the group members.

3. Each member i publishes $g^{\nu_{i,1}}$ making it available only to the group members.

4. Each member i checks if $g^{\theta_1} \stackrel{?}{=} \prod_{j=1}^n g^{\nu_{j,1}}$, where failure (inequality) means that the pad binding parameter and the individual pads do not agree. In this case, since every member publishes its $g^{\nu_{j,1}}$, it is possible to find exactly which member's pad does not agree without knowing the actual value of the pad.

We note that similar testing can be done for the θ generation stage. We omit that due to space limitation.

CONCLUSIONS AND FUTURE WORK

We presented a distributed key generation scheme that allows a pre-specified number of members to jointly generate and update a shared key. We showed that it is possible to make use of the distributed nature of the group—through the use of non-cryptographic techniques—to securely generate and distribute (in the sense of computational security) the future keys. This is achieved by parameterizing the distributed group with a time-varying quantity that is computed at each key update. The parameter binds the members' dynamic one-time pads such that, without knowledge of this parameter, it is not possible to generate a valid one-time pad and, hence, a valid fractional key. In other words, the members' fractional keys are *mixed* or *hidden* with these time-varying pads that implicitly depend on the time-varying group parameter. Hence, even if the *hidden fractional keys* are obtained by an attacker, in the absence of the time-varying group parameter, the attacker does not have immediate access to the group key. Thus, this approach increases group security in a non-cryptographical manner.

We also showed how the group can be initialized with or without an external entity, and still recover the fractional keys of a failed node without an external entity. In developing our methods, we were able to provide some VSS features to verify that the group parameter, and the member generated secrets, are indeed related.

More importantly, the group shared keys are generated using a strong one-way function and hence the loss of the shared key at a particular time interval compromises *neither* the integrity of the future keys *nor* the integrity of the past keys.

REFERENCES

[1] R. Canetti, B. Pinkas, "A taxonomy of multicast security issues", INTERNET-DRAFT, May 1998.

[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] R. Poovendran, S. Ahmed, S. Corson, J. Baras, "A Scalable Extension of Group Key Management Protocol", *Proc. of 2nd Annual ATIRP Conference*, 187-191, Feb. 2-6, 1998, Maryland.

[6] M. Bellare and S. Micali, "Non-Interactive Oblivious Transfer and Applications", *Advances in Cryptology - Crypto '89*, Springer-Verlag, 547-557.

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

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

[9] G. J. Simmons, "An Introduction to Shared Secret and/or Shared Control Schemes and Their Applications", G. J. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, 441-497, IEEE Press, 1992.