Admission Control in Peer-to-Peer: Design and Performance Evaluation

Nitesh Saxena
Computer Science Dept.
UC Irvine
nitesh@ics.uci.edu

Gene Tsudik
Computer Science Dept.
UC Irvine
gts@ics.uci.edu

Jeong Hyun Yi
Computer Science Dept.
UC Irvine
jhyi@ics.uci.edu

ABSTRACT

Peer-to-Peer (P2P) applications and services are very common in today's computing. The popularity of the P2P paradigm prompts the need for specialized security services which makes P2P security an important and challenging research topic. Most prior work in P2P security focused on authentication, key management and secure communication. However, an important pre-requisite for many P2P security services is secure admission, or how one becomes a *peer* in a P2P setting. This issue has been heretofore largely untouched.

This paper builds upon some recent work [11] which constructed a peer group admission control framework based on different policies and corresponding cryptographic techniques. Our central goal is to assess the practicality of these techniques. To this end, we construct and evaluate concrete P2P admission mechanisms based on various cryptographic techniques. Although our analysis focuses primarily on performance, we also consider other important features, such as: anonymity, unlinkability and accountability. Among other things, our experimental results demonstrate that, unfortunately, advanced cryptographic constructs (such as verifiable threshold signatures) are not yet ready for prime time.

Categories and Subject Descriptors

C.2 [**Computer-Communications Networks**]: Security and Protection; C.2.2 [**Network Protocols**]: Applications; C.4 [**Performance of Systems**]: Design Studies

General Terms

Security, Performance

Keywords

peer-to-peer, security, group membership, access control, mobile ad-hoc networks, admission control

Proceedings of the f[t] ACM Workshop Security of Ad Hoc and Sensor Networks Fairfax, Virginia

.

1. INTRODUCTION

In a typical P2P scenario, a number of distributed entities (peers) perform some function in a decentralized manner. P2P applications are generally divided into three classes: parallelizable, content-based and collaborative. A *parallelizable* P2P application splits a large task into smaller pieces that execute in parallel at a number of independent peer nodes. A *content-based* P2P application focuses on storing information at various peers in the network. A *collaborative* P2P application allows peers to collaborate, in real time, without relying on central servers to collect and relay information. Current P2P settings are clearly very diverse. For example, some collaborative P2P applications require synchronous operation, while others, such as content-based, operate in a disconnected, asynchronous manner. P2P communication models vary as well: from one-to-many or few-to-many to many-to-many.

The rising popularity of P2P applications prompts the need for specialized P2P security services and mechanisms. This need has been recognized by the research community. However, the bulk of prior work has been in the context of authentication, anonymity, key management and, in general, secure P2P communication. Although these are certainly important, another equally important topic has remained mostly unaddressed. Informally, it has to do with how one becomes a peer in a P2P paradigm. More concretely, the technology for secure admission of peers into a P2P application **simply does not exist**. This statement does not contradict the fact that there are currently many operating P2P applications; they either operate in a completely open manner (i.e., have no admission control whatsoever) or admit peers on some *ad hoc* basis.

In this work we focus on admission control mechanisms for P2P settings. Although some P2P applications have behind-the-scenes servers (e.g., Naptster), many operate as true *peer* groups with a flat structure (no hierarchy) where all peer nodes have identical rights and responsibilities. It is in such "server-less" P2P environments that admission of new members is a real challenge.

The rest of the paper is organized as follows: Section 2 provides some background on P2P systems. Section 3 presents our system model and discusses design challenges in P2P admission control. Next, Section 4 describes the general P2P admission process. Then, Section 5 discusses the design and evaluation of the actual admission control protocols. Experimental results are presented and analyzed in Sections 6 and 7, respectively. Finally, related work is overviewed in Section 8. (Appendix A contains detailed protocol descriptions).

2. BACKGROUND

There is a wide variety of P2P systems currently in operation. They include unstructured systems such as Gnutella [31] and Freenet [3] where peers are unaware of the total membership of nodes in

the system. In these systems, a request is recursively flooded or randomly forwarded to directly connected peers until it is either answered or its hop count expires. More structured P2P systems (e.g., CAN [24], Chord [30], Pastry [26], and Tapestry [32]) tend to use some form of a distributed hash table (DHT) which guarantees that desired content is always found, if it exists.

Most large-scale P2P systems are asynchronous in nature, i.e., constant on-line presence is not assumed or required from peer nodes. However, one extreme of P2P systems covers more traditional, synchronous group communication systems such as Spread [29], Totem [1] or Horus [25] where scalability is typically limited, membership awareness is total and group membership requires constant on-line presence of each peer.

Nodes in a P2P system can use a wide spectrum of communication paradigms. At one end of the spectrum are powerful workstations connected to stable, high-speed, wired networks. At the other end are small wireless devices (such as PDAs) forming mobile ad hoc networks (MANET-s). As mentioned earlier, the goal of this paper is to explore admission control mechanisms for P2P systems. In doing so, we try to be as general as possible and stay independent of any specifics of the underlying P2P system.

3. SECURITY

Since P2P systems are usually deployed over open networks, security is obviously a very important concern. We identify three main pillars of P2P security: key management, trust management and access control. The purpose of key management is to establish cryptographic keys to allow secure communication among the members. Trust management aims to reduce, or possibly block, the spread of malicious content and assure reliability as well as availability of service.

Access control essentially amounts to controlling membership. It is easy to see that both key management and trust management are effective **only after** a member is allowed to join (access) the group. Without a secure admission process (which, at the very least, should include the authentication of the prospective member), we argue that there is no point in using secure key management and trust management since a malicious prospective member can easily generate any number of false identities. For this reason, our focus is on P2P admission control.

3.1 System Model

Our system model closely follows the admission control framework developed by Kim, et al. in [11]. This framework classifies group admission policy according to the entity (or entities) charged with admission decisions. The classification includes simple admission control policies, such as static ACL-based admission, as well as admission based on the decision of some fixed entity; either external (e.g., a TTP) or internal (e.g., a group founder). These simple policies are relatively easy to support and do not present much of a challenge. An in-depth discussion of various issues surrounding admission control policies can be found in [11].

Things become much more interesting when the peers themselves are responsible for admitting new members. In this case, admission is typically based on some form of limited consensus among current peers. Limited consensus is essentially equivalent to attaining a threshold (or a minimum number) of current members who agree to admit a prospective peer to their group.

As in [11], we distinguish between admission policies based on fixed and dynamic thresholds. A fixed threshold is specified as the minimum number of votes, whereas, a dynamic threshold is specified as a fraction of the current group size. A fixed threshold is essentially a t-out-of-n model where the threshold t is fixed and n (current group size) varies over time. In contrast, with a dynamic threshold (such as 30%), t shrinks or grows in tandem with n.

3.2 Population Size and Awareness

Since the population of peers in most P2P systems is dynamic, i.e., peers join and leave the group at any time, establishing precise current group size can be problematic. However, especially for dynamic thresholds, it is imperative to determine the current group size. (For fixed thresholds, it is only important to establish whether the current group size is less than the threshold, i.e., $n < t$. If so, special "below-threshold" policy must be used to admit new peers.) This is a challenging problem, especially in a completely distributed, asynchronous and decentralized P2P setting.

Closely related to group size is the issue of membership awareness, i.e., knowing the identities of current peers. This usually requires synchronous, on-line operation. However, synchronous P2P systems are not common (unless one considers group communication systems which do not scale to Internet size). On the other hand, as will be seen later in the paper, synchronous operation makes it relatively easy to conduct certain complicated admission procedures which are unworkable in an asynchronous setting.

In a more common, asynchronous P2P setting, one simple way to solve the group size problem is by imposing or assuming a trusted authority charged with maintaining up-to-date membership information. (In [11], it is referred to as the Group Authority or GAUTH; we use the same terminology here). Every peer can be required to periodically send an authenticated *heart-beat* message to GAUTH which aids in maintaining up-to-date membership information. This clearly violates the peer nature of the system. On other hand, in most current P2P systems [31, 3, 24, 30, 26, 32], a prospective peer must contact a specific node, called a *bootstrapping node*, which provides information about active peers currently on-line. Without trusting this node, it seems that we cannot bootstrap any meaningful service. Hence, some form of trusted authority is required to (at least) maintain current group size information.

We stress that GAUTH is only trusted to keep track of membership information; it is not privy to any peer's or group's secrets. We also recognize that GAUTH represents a potential single point of failure; an adversary can launch denial-of-service attacks against GAUTH and disrupt service. (Of course, replication methods can be used to reduce exposure and improve availability.). Note that rendering GAUTH unavailable only impacts admission of new peers; it **does not** disrupt service as far as the normal operation of the P2P system.

3.3 Certificates

In most modern P2P systems, the identity of each peer is determined by some unique identifier (e.g., an IP address or a DNS name) or by a hash thereof. Neither approach is secure since an adversary can always choose a set of node IDs that are not random, or choose a set of IP addresses that hash to a desired range in the node ID space. This is known as a Sybil attack [5]. It is particularly inappropriate to use the node ID as a function of the IP address in a P2P system with highly dynamic peer population.

In order Sybil attacks, we can use the public key of the node to calculate the node ID. This is contingent upon every peer having a standard (e.g., X.509v3) public key certificate (PKC) issued by a recognized Certification Authority (CA) [9]. In our model, rather than using standard long-term identity certificates, a special-purpose certificate asserting group membership is issued to each peer upon admission. We refer to it as a *Group Membership Cer-*

tificate (GMC).

A GMC is in many respects similar to a standard PKC. It includes the name of the group and the usual fields, such as issuance time and the validity interval. A GMC may reference the particular member's identity PKC. Alternatively, or additionally, it may also contain a distinct public key (for which the member knows the corresponding secret key) to be used as the member's unique security credential within the group. A peer member can prove membership by supplying a valid GMC and demonstrating knowledge (e.g., by signing a message) of the private key corresponding to the public key referred to, or included in, the GMC.

4. ADMISSION CONTROL

In this section we provide a more detailed discussion of admission control. The table below summarizes the notation used in the rest of the paper.

Table 1: Notation Summary

GAUTH	group authority
n	total number of peers
t	threshold ($t \leq n$)
i, j	indices of peers $0 < i, j \leq n$
M_i, M_j	i^{th} and j^{th} peer, respectively
id_i	identifier of M_i
PKC_i	public key certificate of M_i
GMC_i	group membership certificate of M_i
$SK_i,\ PK_i$	M_i's secret and public keys, respectively
GSK	group secret key
$S_i(x)$	signature of message x generated with SK_i
ss_i	secret share of M_i
$pss_j(i)$	partial secret share for M_i by M_j
SL_i	signers list

4.1 Concepts

We now describe a generic peer-based admission process.

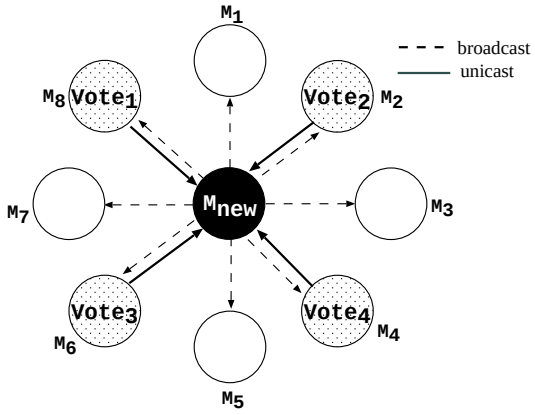


Figure 1: Admission Control

Step 0. Bootstrapping: A prospective peer M_{new} obtains the *group charter* [11] out of band and then the information of current group size from either GAUTH or some bootstrap node. The group charter is a signed document containing admission policies and various parameters such as group name, signature/encryption algorithm identifiers, threshold type (fixed or dynamic), and other optional fields. This process is performed only once per admission.

Step 1. Join Request: M_{new} initiates the protocol by sending a join request (JOIN_REQ) message to the group. This message, signed by M_{new}, includes M_{new}'s public key certificate (PKC_{new})[1] and the target group name. How this request is sent to the group is application-dependent.

Step 2. Admission Decision: Upon receipt of JOIN_REQ, a group member first extracts the sender's PKC_{new} and verifies the signature. If a voting peer approves of admission it replies with a signed message (JOIN_REP). Several signature schemes (described in section 5) can be used for this purpose. Then, M_{new} collects and verifies at least t such votes.

Note that, depending on the underlying signature scheme, this step may involve multiple rounds and co-ordination among signing members.

Step 3. GMC Issuance: Exactly who issues the GMC_{new} for M_{new} depends on the security policy. If the policy stipulates using an existing GAUTH , once enough votes are collected (according to the group charter), M_{new} sends GAUTH a group membership certificate request message (GMC_REQ). It contains: PKC_{new}, group name, and the set of votes collected in Step 2. In a distributed setting with no GAUTH , M_{new} verifies the individual votes, and, from them, composes its own GMC_{new}.

Armed with a GMC, M_{new} can act as a *bona fide* group member. To prove membership to another party (within or outside the group) M_{new} simply signs a message (challenge) to that effect.

4.2 System Design

The admission control system is made up of four components; policy manager, group membership controller, libraries for various signature schemes, and signature format processor. Figure 2 shows the architecture.

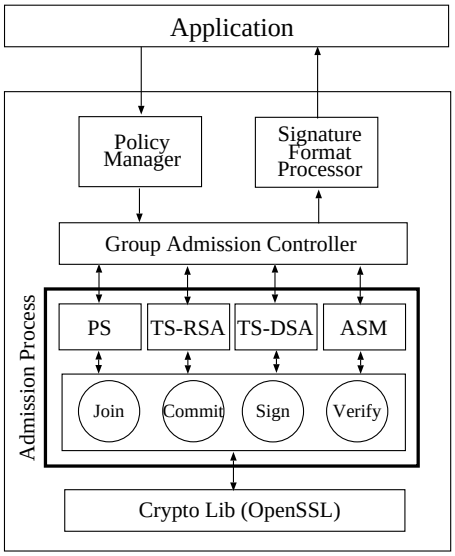


Figure 2: Group Admission Control System

A *policy manager* is the component which checks for conformance to the policy specified in the group charter. First, the policy manager checks the threshold type. If the threshold type is static, it checks if the number of current members is at least equal to the

[1]Note that PKC_{new} does not have to be an identity certificate; it could also be a group membership certificate for another group.

threshold ($n \geq t$). If $n < t$, the policy manager enforces the `BelowThreshold` policy which requires it to either forward the `JOIN_REQ` to GAUTH directly, or to reset the threshold to reflect current n.

In some P2P systems (especially in a MANET), group size can fluctuate drastically within a short time. As the number of peers grows or shrinks, we need to increase or decrease the threshold. Since updating the threshold is an expensive operation, it is impractical for every membership event to trigger an update process. In order to prevent this, we apply a simple *window* mechanism. Specifically, every member keeps state of n_{old}, which is the group size at the time of the last threshold-update process. A new threshold-update process is triggered only when the difference between the current group size n_{cur} and n_{old} is greater than Win – the window buffer. In other words, threshold update process is triggered only when $|n_{cur} - n_{old}| \geq Win$.

The *group admission controller* includes functions used to identify admission protocols and distribute the messages to the corresponding libraries. Four different *admission control libraries* built atop OpenSSL [19] form the core cryptographic engine of the system. The details of message handling are described in the following section. The *signature format processor* transforms all signatures into the standard format. That is, membership certificates (GMC-s) generated by our system are compatible with X.509v3 [9], and signatures on all protocol messages are PKCS7-formatted [23].

5. PROTOCOL DESIGN

In this section, we describe the protocols used for making admission decision for both centralized and decentralized models. (We assume existence of GAUTH in all centralized protocols). A more detailed description of each protocol can be found in **Appendix A**.

5.1 Plain Signatures

Plain digital signature schemes, such as RSA and DSA (denoted collectively as PS), are natural and default candidates for admission control. Although we use RSA in our protocol examples below, just about any signature scheme can be used instead. Figures 3 and 4 illustrate the centralized and the decentralized RSA protocols, respectively.

$M_{new} \rightarrow M_i$:	$m_1, S_{new}(m_1), PKC_{new}$	(1)
$M_{new} \leftarrow M_i$:	$vote_i, GMC_i$	(2)
$M_{new} \rightarrow$ GAUTH :	$\{(vote_1, \cdots, vote_t)\}$, $SL_{new}, m_2, S_{new}(m_2)$	(3)
$M_{new} \leftarrow$ GAUTH :	GMC_{new}	(4)

m_1=`JOIN_REQ`, m_2=$\texttt{GMC\_REQ}_{new}$
$vote_i = (m_1 || id_{new})^{SK_i} \mod n_i$

Figure 3: Centralized RSA Protocol

$M_{new} \rightarrow M_i$:	$m, S_{new}(m), PKC_{new}$	(1)
$M_{new} \leftarrow M_i$:	$vote_i, GMC_i$	(2)

m=$\texttt{GMC\_REQ}_{new}$
$vote_i = (m)^{SK_i} \mod n_i$

Figure 4: Decentralized RSA Protocol

An important advantage of plain signatures is that each current member can sign the join request *asynchronously*, i.e., the admission process does not require coordination among current members. The "applicant" (M_{new}) can approach each member at its own leisure and gradually collect the required number of votes. The main drawback is the need to keep a linear number of votes before approaching the GAUTH . Also, a separate signature verification for each vote can amount to significant overhead.

5.2 Threshold RSA

Recently, Kong, et al. [13, 14, 12] proposed the use of threshold signature schemes for distributing the functions of a certification authority throughout a MANET. They suggested an RSA threshold signature scheme. We will refer to it as TS-RSA. In this section, we describe their scheme and apply it to admission control in P2P systems.

During initialization, a trusted dealer (i.e., a GAUTH or a bootstrapping node in our context) is involved in generating an RSA modulus N which will be common for the entire group, a secret function $f(x)$ such that $f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} \pmod N$, where a_0 is a group secret key GSK. The dealer distributes a secret share $ss_i = f(i)$ to group founders M_i-s. After that, the dealer is no longer required.

Since there may be compromised members who can generate false shares and false signatures thereafter, the dealer provides a *witness* of $f(x)$ which is represented by $\{g^{a_0}, g^{a_1}, \cdots, g^{a_{t-1}}\} \pmod N$ for a certain $g \in \mathbb{Z}_n^*$, and publishes it for Verifiable Secret Sharing (VSS) [22].

$M_{new} \rightarrow M_i$:	$m_1, S_{new}(m_1), PKC_{new}$,	(1)
$M_{new} \leftarrow M_i$:	GMC_i	(2)
$M_{new} \rightarrow M_j$:	SL_{new}	(3)
$M_{new} \leftarrow M_j$:	$vote_j, pss_j(new)$	(4)
$M_{new} \rightarrow$ GAUTH :	$m_1, \{vote_1, \cdots, vote_t\}$, $SL_{new}, m_2, S_{new}(m_2)$	(5)
$M_{new} \leftarrow$ GAUTH :	GMC_{new}	(6)

m_1=`JOIN_REQ`, m_2=$\texttt{GMC\_REQ}_{new}$
$vote_j = (m_1 || id_{new})^{d_j} \mod N$

Figure 5: Centralized TS-RSA Protocol

$M_{new} \rightarrow M_i$:	$m, S_{new}(m), PKC_{new}$	(1)
$M_{new} \leftarrow M_i$:	GMC_i	(2)
$M_{new} \rightarrow M_j$:	SL_{new}	(3)
$M_{new} \leftarrow M_j$:	$vote_j, pss_j(new)$	(4)

m=$\texttt{GMC\_REQ}_{new}$
$vote_j = (m)^{d_j} \mod N$

Figure 6: Decentralized TS-RSA Protocol

This scheme can be viewed as an ideal solution for peer group admission control, mainly because it offers minimal dealer involvement. However, it has a major flaw in checking correctness of the secret share ss_{new} after summing up all $pss_j(new)$. In other words, it does not provide verifiability of secret shares [16].

As a result, malicious or compromised users can send fake shares to new members without being detected. This limits the scheme's applicability in providing security services in a dynamic group setting. Another drawback of this scheme (albeit, a relatively minor one) is that it requires a trusted dealer to generate the group secret (RSA key) and share it among the initial members, although presence of the dealer is limited to a short period of time (bootstrapping). Furthermore, in order to maintain the secrecy of ss_j-s, we need to use the *random shuffling* technique proposed in [13].

5.3 Threshold DSA

In this section, we consider the scheme using threshold DSA signatures, denoted as TS-DSA. This is an extension of the scheme in [6] where n, the number of group members, can be increased without changing the group secret.

Unlike TS-RSA, TS-DSA can be initialized by a group of t or more founding members using Joint Secret Sharing (JSS) [20] as well as a dealer since a secret polynomial is selected over a *public* prime modulus q. Thus, TS-DSA does not require the dealer even for the bootstrapping phase.

We assume that the centralized protocol requires a dealer (e.g., GAUTH) to set up the polynomial over the group secret x, and, in the decentralized version, the secret polynomial is *collectively* generated by group founding members $M_i(|i| \geq t)$. We apply JSS once again to select a random secret k in both centralized and decentralized versions. For more details, please refer to [20].

$M_{new} \rightarrow M_i$: $m_1, S_{new}(m_1), PKC_{new}$, (1)
$M_{new} \leftarrow M_i$: GMC_i (2)
$M_{new} \rightarrow M_j$: SL_{new} (3)
$M_{new} \leftarrow M_j$: $u_j,\ v_j$ (4)
$M_{new} \rightarrow M_j$: R (5)
$M_{new} \leftarrow M_j$: $vote_j, pss_j(new)$ (6)
$M_{new} \rightarrow$ GAUTH : $m_1, \{vote_1, \cdots, vote_t\}, R, SL_{new}, m_2, S_{new}(m_2)$ (7)
$M_{new} \leftarrow$ GAUTH : GMC_{new} (8)

m_1=JOIN_REQ, m_2=GMC_REQ_{new}
$vote_j = k_j((m_1||id_{new}) + x_j R) \mod q$

Figure 7: Centralized TS-DSA Protocol

$M_{new} \rightarrow M_i$: $m, S_{new}(m), PKC_{new}$ (1)
$M_{new} \leftarrow M_i$: GMC_i (2)
$M_{new} \rightarrow M_j$: SL_{new} (3)
$M_{new} \leftarrow M_j$: $u_j,\ v_j$ (4)
$M_{new} \rightarrow M_j$: R (5)
$M_{new} \leftarrow M_j$: $vote_j, pss_j(new)$ (6)

m=GMC_REQ_{new}
$vote_j = k_j(m + x_j R) \mod q$

Figure 8: Decentralized TS-DSA Protocol

This scheme provides verifiability of secret shares. However, just like in TS-RSA, random shuffling is required to securely transfer $pss_j(new)$. Similar to [6], this scheme is secure only if there are less than $t' = \lfloor \frac{t+1}{2} \rfloor$ malicious (subverted) members. To generate a random secret without the dealer, extra $O(n^2)$ communication rounds are required.

5.4 Accountable Subgroup Multisignatures

Ohta, et al. [17] proposed this interesting scheme, denoted as ASM, which enables any subgroup of a given group of potential signers, to sign in a way that the signature, while of constant length, provably reveals the identities of all individual signers to any verifier. This scheme takes advantage of the homomorphic property of Schnorr signatures [27] to construct an efficient ASM scheme.

$M_{new} \rightarrow M_i$: $m_1, S_{new}(m_1), PKC_{new}$, (1)
$M_{new} \leftarrow M_i$: c_i, GMC_i (2)
$M_{new} \rightarrow M_j$: E, C, SL_{new} (3)
$M_{new} \leftarrow M_j$: $vote_j$ (4)
$M_{new} \rightarrow$ GAUTH : $m_1, \{vote_1, \cdots, vote_t\}, C, SL_{new}, m_2, S_{new}(m_2)$ (5)
$M_{new} \leftarrow$ GAUTH : GMC_{new} (6)

m_1=JOIN_REQ, m_2=GMC_REQ_{new}
$vote_j = r + x_j(E||id_{new}) \mod q$

Figure 9: Centralized ASM Protocol

The verification phase in this scheme requires only two modular exponentiations and k modular multiplications, since an ASM

$M_{new} \rightarrow M_i$: $m, S_{new}(m), PKC_{new}$ (1)
$M_{new} \leftarrow M_i$: c_i, GMC_i (2)
$M_{new} \rightarrow M_j$: E, C, SL_{new} (3)
$M_{new} \leftarrow M_j$: $vote_j$ (4)

m=GMC_REQ_{new}
$vote_j = r + x_j E \mod q$

Figure 10: Decentralized ASM Protocol

is effectively the same as a regular Schnorr signature. This is very efficient since, otherwise, t verifications would be performed. Unlike threshold signatures, no dealer is assumed. Another notable feature of ASM is full accountability of signers. However, since each signer's identity should be included, the length of a signature is linear in the number of signers.

5.5 Comparison

The protocols discussed thus far offer very different alternatives for P2P admission control. Table 2 summarizes their key features.

PS relies on each signer having its own key-pair which is independently generated. The same independence of key generation applies to ASM schemes. In contrast, TS-RSA and TS-DSA entail a complicated setup phase and a non-trivial join procedure with multiple protocol rounds. Moreover, the complex setup does not eliminate the need for having specific key material for signing messages within the group. Between threshold schemes, TS-DSA differs from TS-RSA in that the group secret key can be generated by a group of members without the dealer.

PS is the most general in that neither on-line presence of all signers nor membership awareness is necessary. This is important for asynchronous, off-line groups such as content sharing communities.

PS and ASM directly identify the signers. Signers' accountability is impossible (or at least hard to attain) with TS-RSA and TS-DSA. We remark that accountability is not always desired. In contrast, TS-RSA and TS-DSA provide the anonymity of the signers.

TS-RSA, TS-DSA, and ASM have only one resulting signature, while, in PS, the number of signature is linear in the number of signers. Similarly, both TS-RSA and TS-DSA have a constant length of the final signature independent of the number of signers. The signature size in PS depends on the number of signers. Since ASM contains the identity of signers, the signature may become larger as the group grows.

Table 2: Feature Summary

Key Features	PS	TS-RSA	TS-DSA	ASM
Dealer involved		√		
On-line presence		√	√	√
Accountability	√			√
Anonymity		√	√	

6. PERFORMANCE

We implemented the group admission control toolkit as described in 4.2. It consists of the cryptographic functions which are developed using the OpenSSL library [19]. This toolkit is written in C for Linux, and currently consists of about 45,000 lines of code. The source code for the admission control toolkit is available in [21].

We integrated our admission control system with both Secure Spread [28] and Gnutella [7] in order to measure the performance in the context of *real* P2P applications. Secure Spread is selected

as an example of a synchronous P2P system, and Gnutella as an asynchronous one.

We measured the basic operations and then compared the performance of four cryptographic protocols with both fixed and dynamic thresholds. We used 1024-bit modulus in all mechanisms; that is, 1024-bit N in RSA and TS-RSA, and 1024-bit p and 160-bit q in TS-DSA and ASM[2].

Since each protocol has different number of communication rounds, we measured total processing time from sending of the `JOIN_REQ` to obtaining new GMCs[3]. This means the join cost includes not only the signature generation and verification time in basic operations, but also the communication costs such as packet encoding/decoding time, the network delay, and so on. To get reasonably correct results, the experiments were repeated 1000 times for each.

6.1 Basic Operation Costs

In this section, we demonstrate the cost of each signature scheme used as a primitive in our admission control mechanisms. Table 3 shows, for each scheme, the number of exponentiations for signature generation and signature size which resulted from t partial signatures. With regard to the number of exponentiation, TS-DSA takes one extra exponentiation order to get r without knowledge of k or k^{-1} in a distributed fashion. The length of the signature in PS and TS-RSA is linear in t. Note that, in both PS and ASM, the combined signatures include the signers' identities and the total signature size grows accordingly. TS-DSA has a constant signature size.

Table 3: Signing Cost Analysis ($|K|$: key length, $|id|$: length of signer's id, unit: bit)

	No of Expos.	Signature size
PS	t	$t * (\|K\| + \|id\|)$
TS-RSA	t	$\|K\|$
TS-DSA	$t+1$	$2 * \|q\|$
ASM	t	$\|q\| + \|E\| + (t * \|id\|)$

Table 4 shows our measurements for signature generation with plain RSA, DSA and Schnorr schemes which are form basis of TS-RSA, TS-DSA, and ASM, respectively. Table 5 shows the cost of signature generation in terms of key size, where t=3. In TS-RSA, the cost in generating a partial signature is almost the same as that of RSA signature generation, but we need extra cost to compute t times *Lagrange coefficient* for partial private key. Similarly, TS-RSA is slightly better than TS-DSA with 512-bit modulus, while TS-DSA is faster than TS-RSA with larger key size. As evident from the table, ASM is the best performer.

Table 4: Signature Generation Costs of Basic Schemes in msecs (P3-977MHz)

Key	RSA	DSA	Schnorr
512	1.390	1.440	1.650
768	3.940	2.680	2.775
1024	6.820	3.815	4.136
2048	41.880	12.290	12.830

Table 6 shows the measured signature verification costs for all aforementioned signatures. Table 7 shows the cost of signature verification in terms of key size. In PS, the cost of signature verification is proportional to the threshold. All other schemes, except PS, have only one resulting signature due to the aggregation of partial signatures. It is easy to see that the verification costs of TS-DSA and ASM are similar to that of underlying standard signatures. However, verification cost for TS-RSA is extremely high. This is because the computation of $m^N \pmod N$ in *t-bounded offsetting algorithm* [13] has to be done almost every time the signature is verified.[4] Unlike plain RSA, we can not apply the *Chinese Remainder Theorem* to speed up computation, since the factors of N are unknown to the verifier. From our experiments, we found that this computation takes more than 95% of the total verification cost. This is a critical observation. Contrary to our intuition, the result shows that TS-RSA is much worse than TS-DSA in terms of signature verification cost.

Table 5: Signature Generation Costs in msecs (t=3, P3-997MHz)

Key	PS	TS-RSA	TS-DSA	ASM
512	3.710	15.920	23.730	7.930
768	14.010	38.349	32.380	11.300
1024	24.220	70.429	40.920	15.902
2048	128.500	425.176	104.420	41.760

Table 6: Signature Verification Costs of Basic Schemes in msecs (P3-977MHz)

Key	RSA	DSA	Schnorr
512	0.198	1.910	1.940
768	0.238	3.430	3.450
1024	0.287	4.755	4.774
2048	0.604	15.150	15.030

Table 7: Signature Verification Costs in msecs (t=3, P3-997MHz)

Key	PS	TS-RSA	TS-DSA	ASM
512	0.465	5.010	2.090	1.9400
768	0.723	12.550	3.950	4.025
1024	0.763	24.410	5.020	5.290
2048	1.780	144.460	15.450	15.760

6.2 Secure Spread Experiments

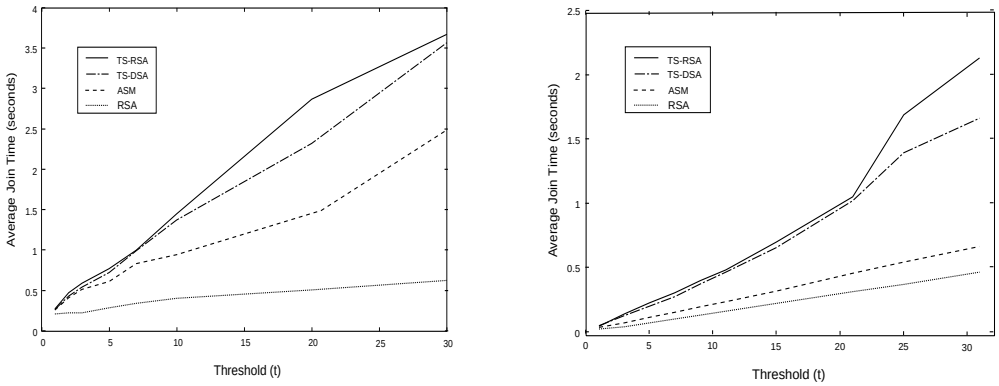
Secure Spread [28] is an application built on top of Spread [29] – a reliable wide-area group communication system. Secure Spread adds group key management and secure communication services to Spread. In its present form, Secure Spread supports only static group access control. This is done via the *flush* mechanism which is used by each current group member to acknowledge every membership change (e.g., join, leave, partition, merge). A prospective member can join a group only after it has received *flush OK* messages from all current group members. However, this mechanism offers no security, since there involves no authentication of either new or current members. Moreover, all group members must be involved in each admission process.

We integrated our admission control mechanisms with Secure Spread. The integration involved extensions to Spread API. All

[2]DSA key of 1024 bits is a little more secure than 1024-bit RSA key, because the RSA key is a composite number and the DSA key is a prime [10].

[3]In these experiments we did not consider the *partial share shuffling* for both TS-RSA and TS-DSA.

[4]We ran each test case 1000 times. We observed that with SHA-1 as the hash function, about 50% of trials underwent this operation in case of 512-bit key and t=2. For all other test cases, it occured almost all the time.



(a) Secure Spread

(b) Gnutella

Figure 11: Fixed Threshold Cost

group admission protocol messages (e.g., `JOIN_REQ`, `JOIN_REP`, etc.) are sent encapsulated within standard Spread messages. Spread unicasting and muticasting are used to send such messages. Once the joining member receives its GMC, it becomes the new member of the group and the group key is updated.

For our experiments with Secure Spread, we used a cluster of 10 machines at Johns Hopkins University. Each machine has P3-667 MHz CPU, 256 KB Cache and 256 MB memory and runs Linux 2.4. We ran Spread daemons on all machines which formed a Spread Machine Group. Almost equal number of clients running on these machines connect randomly to the daemons. The new joining member is a client running on a machine at UC Irvine with a Celeron 1.7 GHz CPU, 20 KB cache and 256 MB memory.

Experiments were performed with the above testbed for both fixed and dynamic thresholds for all signature schemes discussed thus far.

Figure 11(a) shows the plot for the average time taken by a new member to join a group with a fixed threshold. We performed this test with 4-5 processes on each machine and measured the join cost by changing the threshold. As expected, plain RSA is the best performer in terms of computation time. However, we also see that both TS-RSA and TS-DSA exhibit reasonable costs ($<$ 1 sec.), at least until t=10.

Figures 12(a) and 12(c) show the plots for the average time for a new member to join a group with a dynamic threshold. In this experiment, the threshold ratios (R) is set to 20% and 30% of the current group size, respectively. The actual numeric threshold is determined by multiplying the group size by R. We measured the performance up to $n = 50$.

6.3 Gnutella Experiments

For the asynchronous P2P experiments, we integrated our mechanisms with Gnut-0.4.21 [7] (an open-source Gnutella [31] implementation). At the setup phase of the Gnutella protocol, a connection is established by communicating so-called `ping` and `pong` messages which are based on IP addresses. In order to prevent Sybil attacks [5], we modified our implementation so that the connection is made only if the responder answers with its valid GMC. For this purpose, we specified two new messages: `sping` and `spong`. The `sping` message contains the requester's PKC, and the `spong` message contains the responder's GMC and its signature (to prove possession of its private key). As in the Spread experiment, all group admission protocol messages are encapsulated within standard Gnutella messages.

We performed all measurements on the following Linux machines connected with a high-speed LAN: P4-1.2GHz, P3-977MHz, P3-933MHz, and P3-797MHz.

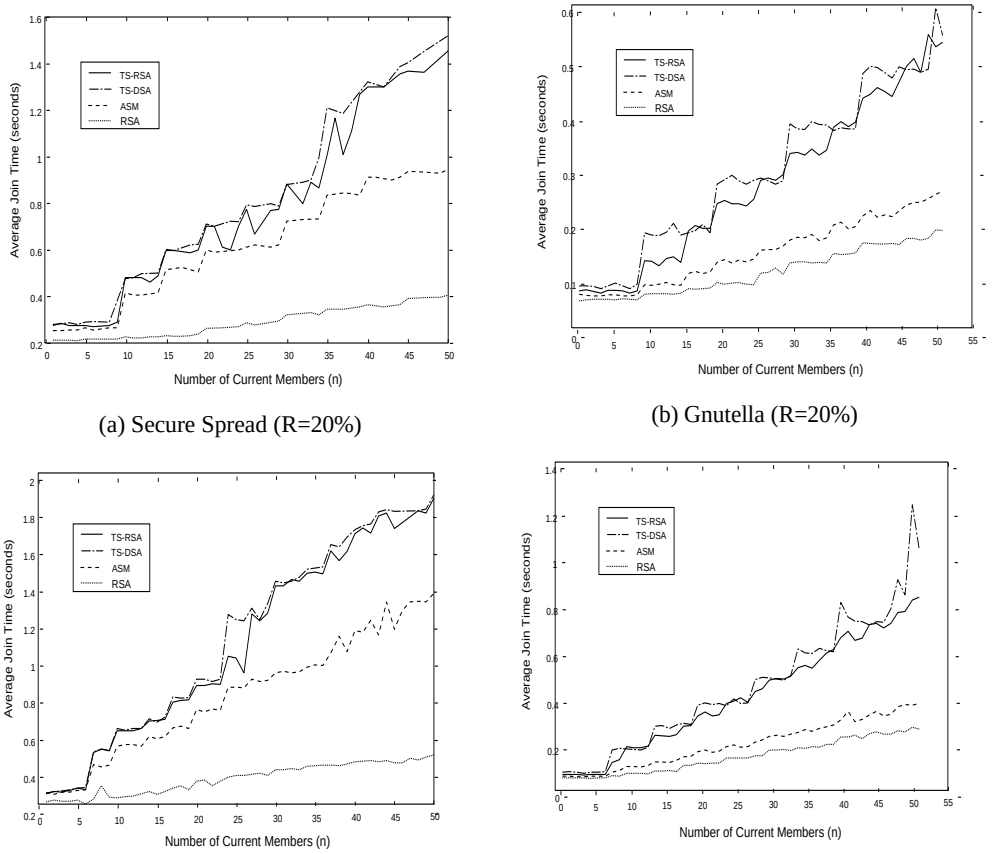
Figure 11(b) shows the join cost for the static threshold case. Finally, Figures 12(b) and 12(d) show the join costs for the dynamic threshold case (20% and 30%, respectively). All of these measurements were performed with the equal number of member processes on each machine.

7. OBSERVATIONS AND DISCUSSION

A few interesting points arise from observing our experimental results. Most observations are quite intuitive. In all tests with both Gnutella and Secure Spread, we note that PS outperforms TS-RSA, TS-DSA and ASM. This is mainly due to the smaller number of rounds in the protocol as well as faster signature generation and verification (refer to Tables 5 and 7). At first, this seems to indicate that advanced cryptographic techniques are not very useful for the purpose of P2P admission control. However, there are some counter-arguments.

The very reason we chose to use the threshold (RSA/DSA) and ASM schemes was because we wanted to reduce the combined signature size in a GMC. As can be seen from Table 3, the signature size (and thus the size of a GMC) in the PS scheme is proportional to the threshold t. This is in contrast with threshold schemes where signature sizes are constant for a given key size. Also, in ASM, the effective signature size is smaller than in the PS, although it includes the identifiers of all signing peers.

For large groups and especially for dynamic thresholds, the GMC will become huge for the PS schemes. This yields a basic tradeoff: PS offers lower join cost but longer GMCs, whereas, other schemes, have shorter (ASM) or constant (TS-DSA/TS-RSA) GMCs but high join cost. The optimal choice would clearly need to depend on the specifics of a particular group. The PS mechanism will be a good choice for small groups or groups of moderate size where bandwidth is not a major concern. For large groups, where joining thresholds are higher and bandwidth is an important issue, threshold and multisignature schemes may yield better results. On the other hand, in power-starved MANET and sensor networks, higher computation costs may still rule out ASM and threshold schemes in favor of PS.



(a) Secure Spread (R=20%)

(b) Gutella (R=20%)

(c) Secure Spread (R=30%)

(d) Gnutella (R=30%)

Figure 12: Dynamic Threshold Cost

We also note that join cost for ASM is better than that for threshold schemes. Also, among threshold schemes, TS-DSA fairs better than TS-RSA for fixed thresholds. ASM and TS-RSA have the same number of rounds, but the difference is in the amount of computation done by the peers (and GAUTH , in case of centralized protocols). In TS-RSA, verification of the threshold signature is extremely high, due to the t-bounded offsetting algorithm mentioned earlier. The costs of signature generation and verification in DSA and Schnorr (on which TS-DSA and ASM are based) as listed in Table 4 and Table 6 are comparable, however, ASM becomes more efficient than TS-DSA because of fewer rounds. For the dynamic threshold case, TS-RSA is slightly ahead of TS-DSA because, in the latter, a considerable amount of time is spent on updating the polynomials as a consequence of the changing threshold. Since the effective signature size is independent of the group threshold, we conclude that the selection among ASM, TS-RSA and TS-DSA schemes has to be made on the basis of the desired features, as discussed in Section 5.5

Another, more transparent, observation is that the graphs follow almost the same pattern for all schemes with both Gnutella and Secure Spread. Not surprisingly, Secure Spread incurs higher costs than Gnutella. This is quite self-explanatory: all communication between the client and the server in Gnutella is point-to-point, whereas, in Spread, two peers communicate via their respective daemons. Also, the communication overhead in Spread is generally higher, due to its synchronous nature and various reliability features.

8. RELATED WORK

With the exception of [11] there appears to be almost no prior work in the area of P2P admission control.

The Antigone [15] project is the closest related work. Antigone includes a flexible framework for secure group communication and utilizes a centralized admission approach geared primarily towards secure multicast scenarios. Antigone offers flexible mechanisms for defining policies about membership, application messages and other aspects.

In Antigone, member admission is mediated by a Session Leader (SL) which interacts with the TTP (that operates on-line) in order to admit a new member. The TTP shares a symmetric key both with the SL and every potential new member. (The TTP acts like a Kerberos AS/TGS). Everyone is expected to know in advance the identity of the SL.

Some of the mechanisms discussed in this paper are akin to limited forms of voting. Electronic voting schemes have been extensively studied starting with the seminal work of Benaloh [2]. Most approaches are based on mix-nets, homomorphic encryption [4] or blind signatures [18].

[8] presents a trust based admission control model which could be integrated with our system to make admission decisions based

on a rating of trust for the prospective member. But, we insist that this scheme can not be used for establishing admission control in its entirety.

9. CONCLUSION

Building upon a peer group admission control framework in [11] we designed several concrete P2P admission control mechanisms based on different cryptographic techniques. We assessed the practicality of these techniques by measuring and analyzing their performance in both synchronous and (more typical) asynchronous P2P settings. The experimental results are slightly disheartening as they cast some doubt upon the practicality of advanced cryptographic techniques, especially, threshold signatures. Moreover, since every scheme has its own pros and cons, we assert that it is impossible to single out one particular scheme for all P2P admission control scenarios.

In summary, although we made some progress towards constructing practical P2P admission control schemes, much remains to be done.

10. REFERENCES

[1] D. Agarwal, L. E. Moser, P. M. Melliar-Smith, and R. K. Budhia. The totem multiple-ring ordering and topology maintenance protocol. *ACM Transactions on Computer Systems*, 16(2):93–132, May 1998.

[2] J. Benaloh. Veriable Secret-Ballot Elections, Yale University PhD thesis. YALEU/DCS/TR-561, 1987.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *ICSI Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[4] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Theory and Application of Cryptographic Techniques*, pages 103–118, 1997.

[5] J. R. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems (IPTPS'02)*, March 2002.

[6] R. Gennaro, S.Jarecki, H.Krawczyk, and T.Rabin. Robust Threshold DSS Signatures. In U. Maurer, editor, *EUROCRYPT '96*, number 1070 in LNCS, pages 354–371. IACR, 1996.

[7] Gnut v0.4.21 source code, `http://schnarff.com/gnutelladev/source/gnut`.

[8] E. Gray, P. O'Connell, C. D. Jensen, S. Weber, J. M. Seigneur, and Y. Chen. Towards a Framework for Assessing Trust-Based Admission Control in Collaborative Ad Hoc Applications. Technical Report TCD-CS-2002-66, Trinity College Dublin, 2002.

[9] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280, IETF, Apr. 2002.

[10] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World (2/e)*. Prentice-Hall. 2002. ISBN 0-13-046019-2.

[11] Y. Kim, D. Mazzocchi, and G. Tsudik. Admission control in peer groups. In *IEEE International Symposium on Network Computing and Applications (NCA)*, Apr. 2003.

[12] J. Kong, H. Luo, K. Xu, D. L. Gu, M. Gerla, and S. Lu. Adaptive Security for Multi-level Ad-hoc Networks. In *Journal of Wireless Communications and Mobile Computing (WCMC)*, volume 2, pages 533–547, 2002.

[13] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for MANET. In *IEEE 9th International Conference on Network Protocols (ICNP)*, 2001.

[14] H. Luo, P. Zerfos, J. Kong, S. Lu, and L. Zhang. Self-securing Ad Hoc Wireless Networks. In *Seventh IEEE Symposium on Computers and Communications (ISCC '02)*, 2002.

[15] P. McDaniel, A. Prakash, and P. Honeyman. Antigone: A flexible framework for secure group communication. In *8th USENIX Security Symposium*, pages 99–114. USENIX, Aug. 1999.

[16] M. Narasimha, G. Tsudik, and J. H. Yi. On the Utility of Distributed Cryptography in P2P and MANETs: the Case of Membership Control. In *IEEE 11th International Conference on Network Protocols (ICNP)*, 2003 (To appear).

[17] K. Ohta, S. Micali, and L. Reyzin. Accountable-subgroup multisignatures. In *ACM Conference on Computer and Communications Security*, pages 245–254, November 2001.

[18] T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols Workshop*, pages 25–35, 1997.

[19] OpenSSL Project, `http://www.openssl.org/`.

[20] T. P. Pedersen. A threshold cryptosystem without a trusted party. In D. Davies, editor, *EUROCRYPT '91*, number 547 in LNCS, pages 552–526. IACR, 1991.

[21] Peer Group Admission Control Project, `http://sconce.ics.uci.edu/gac`.

[22] P.Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Symposium on Foundations of Computer Science (FOCS)*, pages 427–437, 1987.

[23] PKCS #7: Cryptographic Message Syntax Standard, `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/index.html`.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM '01*, pages 161–172, August 2001.

[25] R. V. Renesse, K. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[26] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001.

[27] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[28] Secure Spread Project, `http://www.cnds.jhu.edu/research/group/secure_spread/`.

[29] Spread Project, `http://www.spread.org/`.

[30] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM '01*, pages 149–160, August 2001.

[31] The Gnutella Protocol Specification v0.4, `http://www.clip2.com/GnutellaProtocol04.pdf`.

[32] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

APPENDIX

A. PROTOCOL DESCRIPTIONS

A.1 Centralized RSA

Step 1. A prospective member M_{new} presents its identity certificate PKC_{new} and a signature on JOIN_REQ message m_1 in order to prove the knowledge of the corresponding private key. M_{new} waits for at least t votes from the current group members before it could join the group.

Step 2. The current member M_i signs the message $(m_1||id_{new})$ as a vote using plain RSA scheme and sends it back to M_{new} along with its membership certificate GMC_i.

Step 3. M_{new} verifies at least t $GMC_i (j \in_R i, |j| = t)$, collects t votes, and submits them to GAUTH with GMC_REQ_{new}, and its signature. A GMC_REQ_{new} is a X.509 certificate request [9] which contains an identity of M_{new}, PK_{new}, and other attributes such as a group name. Recall that an identity and a public key of GMC_{new} are bound to those of PKC_{new}. M_{new} also sends a signers' list SL_{new} to the GAUTH so that it can verify the signatures of the respective members.

Step 4. Finally, GAUTH issues GMC_{new} for M_{new} after verifying all votes individually with RSA verification process where id_{new} is derived from GMC_REQ_{new}.

The GAUTH also verifies that the signatures are actually meant for the prospective member with identification id_{new} and not for someone else. This will foil any attempt of a *replay* (one where a new member delivers all its votes to someone else) or a *collusion* (one where two or more prospective members exchange their individual votes) attack.

A.2 Decentralized RSA

Step 1. Same as step (1) in Section A.1 except that the signature is generated on GMC_REQ_{new}.

Step 2. Same as step (2) in A.1 except that the vote is made on GMC_REQ_{new} without id_{new} since GMC_REQ_{new} already contains an identity of M_{new}.

Step 3. Same as step (4) in A.1 except M_{new} issues itself a GMC_{new} which contains $\{\text{GMC\_REQ}_{new}||SL_{new}||S_1||\cdots||S_t\}$, where S_i is $(\text{GMC\_REQ}_{new})^{SK_i} \mod n_i$.

A.3 Centralized TS-RSA

Step 1. Same as step (1) in Section A.1

Step 2. Current members replies with its membership certificate GMC_i to M_{new} after verifying the JOIN_REQ as a join commitment and PKC_{new} therein.

Step 3. M_{new} picks (perhaps at random) t responding $GMC_j (j \in_R i, |j| = t)$ and composes the signer list SL_{new} with the ID of members who have voted. Then, M_{new} sends SL_{new} to each M_j who have sent the join commitments. The reason why we need to send the signer list is to make each M_i know the index of t members in advance to compute *Lagrange coefficient*, denoted by $l_j(x)$, for the partial signatures and shares.

Step 4. Now, each M_j generates $vote_j$ for M_{new} such that $vote_j = (m_1||id_{new})^{d_j} \pmod N$, where a partial private key, d_j, is generated by $ss_j \cdot l_j(0) \pmod N$. Also, M_j computes M_{new}'s partial share $pss_j(new)$ by $ss_j \cdot l_j(new) \pmod N$. The M_i sends back $vote_j, pss_j(new)$ to M_{new}.

Step 5. M_{new} self-initializes its own secret share ss_{new} by summing up all $pss_j(new)$-s. After that, M_{new} requests from GAUTH the membership certificate with the collection of t votes and the signers list.

Step 6. The GAUTH multiplies t votes to generate the signature since $\prod_{j=1}^{t} (m_1||id_{new})^{d_j} \neq (m_1||id_{new})^{d} \pmod N$. To obtain the actual signature, extra procedure, called t-bounded coalition offsetting algorithm [13] is required. If the signature is valid, the GAUTH finally issues the GMC_{new} to M_{new}.

A.4 Decentralized TS-RSA

Step 1. Same as step (1) in Section A.2

Step 2-3. Same as step (2)-(3) in Centralized TS-RSA

Step 4. Same as step (4) in Centralized TS-RSA, except the vote is made on GMC_REQ_{new} without id_{new}.

Step 5. Same as step (6) in Centralized TS-RSA, except M_{new} issues itself a GMC_{new} which contains $\{\text{GMC\_REQ}_{new}||S\}$, where S=$(\text{GMC\_REQ}_{new})^{GSK} \mod N$.

A.5 Centralized TS-DSA

Step 1. Same as step (1) in Section A.1

Step 2. The peers who wish to participate in the admission reply with their respective membership certificates GMC_i to M_{new} after verifying PKC_{new}.

Step 3. M_{new} picks at random t responders $M_j (j \in_R i, |j| = t)$, collects their id_j from their respective GMC-s to form a signer list SL_{new} and sends it to each M_j.

Step 4. Each M_j randomly chooses its polynomial $k_j(z), a_j(z)$ in $\mathbb{Z}_q$ of degree $t' - 1$, where $t' = \lfloor \frac{t+1}{2} \rfloor$. Note that $k(z) = \sum_{j=1}^{t} k_j(z)$, $a(z) = \sum_{j=1}^{t} a_j(z)$. M_j computes $k_j(i)$ and $a_j(i)$ for all signers $M_i (i = 1, \cdots, t)$ in SL_{new}, and then distributes $k_j(i)$ and $a_j(i)$ to all of its co-signers. After receiving her partial shares from the other co-signers, M_j computes k_j and a_j such that $k_j = k(j) = \sum_{l=1}^{t} k_l(j)$, $a_j = a(j) = \sum_{l=1}^{t} a_l(j) \pmod q$. Then, M_j computes u_j and v_j such that $u_j = k_j \cdot a_j \pmod q$, $v_j = g^{a_j} \pmod p$, and sends u_j and v_j back to M_{new}. Why each M_j must take (at least) two polynomials is referred to [6].

Step 5. M_{new} computes R without knowing k and k^{-1} as follows: First, it computes u and v such that $u = \sum_{j=1}^{t} u_j \, l_j(0) \pmod q$ which finally equals to $ka \pmod q$, $v = \prod_{j=1}^{t} (v_j)^{l_j(0)} \pmod p$ which equals $g^a \pmod p$. Next, it computes the inverse $u^{-1} \pmod q$ and finally computes R as $R = (v^{u^{-1}} \mod p) \mod q$ which equals $(g^{k^{-1}} \mod p) \mod q$. Then, M_{new} sends R to M_j.

Step 6. M_j computes $vote_j$ such that $vote_j = k_j((m_1||id_{new}) + x_j R) \mod q$, where x_j is M_j's share of group secret x. Also, M_j computes M_{new}'s partial share $pss_j(new)$ by $x_j \cdot l_j(new) \pmod q$. Then, M_j sends $vote_j$ and $pss_j(new)$ to M_{new}.

Step 7. M_{new} now requests the membership certificate with t votes, a joint challenge R, and the signers list to GAUTH .

Step 8. GAUTH recovers the complete signature S with t votes using Lagrange interpolation and then verifies R and S with standard DSA verification process. If the signature is valid, GAUTH issues the GMC_{new}.

A.6 Decentralized TS-DSA

Step 1. Same as step (1) in Section A.2

Step 2-5. Same as step (2)-(5) in Centralized TS-DSA

Step 6. Same as step (6) in Centralized TS-DSA, except the vote is made on $\texttt{GMC\_REQ}_{new}$ without id_{new}.

Step 7. Same as step (8) in Centralized DSA, except M_{new} issues itself GMC_{new} which contains $\{\texttt{GMC\_REQ}_{new}||R||S\}$, where $R = g^{k^{-1}} \mod p \mod q$, S=$(\texttt{GMC\_REQ}_{new})+xR \mod q$.

A.7 Centralized ASM

Step 1. Same as step (1) in Section A.1

Step 2. To sign a message, each M_i sends a partial commitment c_i such that $c_i = g^{r_i} \pmod p$ to M_{new}.

Step 3. M_{new} multiplies t commitments to obtain the joint challenge E along with all signers information SL_{new} such that $E = H(m_1||C||SL_{new})$, where $C = \prod c_i \mod p$, and then sends E, C and SL_{new} to the t signing members.

Step 4. Each M_j first verifies the hash of E (to confirm that it is signing a membership message and not some arbitrary message from M_{new}). It then computes and returns $vote_j$ such that $vote_j = r_j + x_j(E||id_{new}) \mod q$ back to the M_{new}.

Step 5. M_{new} collects at least t votes and submits them (along with C, SL_{new} and $\texttt{GMC\_REQ}_{new}$) to GAUTH .

Step 6. Finally the GAUTH verifies all votes by checking that $g^S \pmod p$ equals to $C \cdot Y^{(E||id_{new})} \pmod p$, where $S = \sum S_j \pmod q$ and $Y = \prod y_j \pmod p$. If the verification is successful, the GAUTH issues GMC_{new}.

A.8 Decentralized ASM

Step 1. Same as step (1) in Section A.2

Step 2. Same as step (2)-(3) in Centralized ASM.

Step 4. Same as step (4) in Centralized ASM, except the vote is made on $\texttt{GMC\_REQ}_{new}$.

Step 5. Same as step (6) in Centralized ASM, except M_{new} issues itself GMC_{new} which contains $\{\texttt{GMC\_REQ}_{new}||SL_{new}||C||E||S\}$, where $E = H(\texttt{GMC\_REQ}_{new}||\prod g^{r_j}||SL_{new})$, $S = \sum r_j + x_j E$.