Parameterized Access Control: From Design To Prototype*

Ashish Gehani
Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA
ashish.gehani@sri.com

Surendar Chandra
Department of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
surendar@nd.edu

ABSTRACT

Peer-to-peer overlays provide a substrate well suited to building distributed storage systems. Applications that use the infrastructure need the ability to control access to their data. However, traditional authorization services were not designed to operate in the face of network partitions, malicious nodes, and on an Internet-wide scale.

We describe the implementation of the *Decentralized Authentication and Authorization Layer (DAAL)*, a mechanism to leverage the storage functionality of the overlay and obviate the need for an online, centralized access control service. The system can efficiently identify malicious nodes and continue to operate correctly when an arbitrary, predefined fraction of the network is unreachable (as occurs during an attack against the routing infrastructure or during a distributed denial-of-service attack).

DAAL melds the access request efficiency of capability-based systems with the revocation power of reference monitor-based access control lists. It avoids the use of distributed leases as they create a vulnerability window during which there is a gap between the security policy and configuration. Actualizing the design can be challenging. Hence, we describe the protocol details and how they can be abstracted behind a minimal, intuitive application programming interface. As a proof of concept, we implemented DAAL as a Java prototype on a 200-node peer-to-peer overlay distributed across the world.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls, Authentication, Cryptographic controls

1. INTRODUCTION

The volume of data emerging from sensor systems depends on the resolution at which individual sensors monitor their environment. As a result, the improvements in the manufacturing process used to fabricate sensor chips have resulted in greater amounts of digital data being created by a range of systems. These include radar networks, coastal surveillance systems, weather monitoring infrastructures, seismic activity recorders, and astronomical observation systems. Such networks are increasingly being interconnected using the Internet instead of using dedicated lines between the distributed sensor nodes and the locations where the output data is utilized. To minimize the amount of network traffic, such systems are organized as peer-to-peer networks, allowing the consumer of each piece of data to retrieve it directly from the node where it was produced or stored.

Prior distributed access control infrastructures, such as Kerberos [2] for AFS and WebOS's CRISIS [1], usually rely on specialized authentication and authorization servers. However, dependence on centralized online services is not scalable for peer-to-peer environments. An attacker could halt file access operations across the entire system by launching a distributed denial-of-service attack against the security servers. If a revocation server is unable to establish a network connection to a remote node, the rights previously granted can be amplified. A protection mechanism must scale with the size of the system that it serves. Further, the cost of identifying malicious nodes must be minimal or the system will be open to denial of services.

Parameterized access control allows a user to select different properties for each object in the system, depending on whether the user needs capability-like characteristics for an object whose access requests must complete quickly or access control list-like functionality for an object to which access should be efficiently revoked. Previously, we studied [6] how users can trade the efficiency of permission request and revoke operations by adjusting an object's parameters and identified a criterion for parameter selection to guarantee a level of performance given a predefined fraction of malicious peers.

This paper describes how to translate parameterized access control from a high-level design into concrete authorization protocols that can be embedded in the operations of a wide-area storage system. Further, we report on our experience developing a Java prototype and its operation on PlanetLab [14].

2. OVERVIEW

Implementing access control protection requires a means to define subjects, objects and rights [11], methods to create and destroy each of them, and a reference monitor that can intercede on all accesses [8]. DAAL's operation for creating new subjects is done offline using a trusted authority. All the other operations, including access requests and the resulting grants or denials, occur in a completely decentralized manner with an arbitrary subset of peers participating in the process. DAAL can provide rapid access to and revocation of rights for data stored in the overlay. Data objects are kept encrypted and signed with cryptographic capabilities.

The protocol has three stages. The first uses a specific version of public key cryptography to transform these capabilities so that only authorized users can access them. The second splits the transformed capabilities into β fragments using threshold cryptography, such that only a predefined threshold α of them are needed to reconstitute the capability. The third stage distributes the fragments independently in the overlay network itself. This constitutes a grant operation. Access requires this process to be reversed. Revocation is effected by deleting some of the fragments from the overlay.

Using Boneh-Franklin [3] Identity Based Encryption, no online lookups are needed to verify signatures or encrypt data being sent between users. Pedersen's [13] Verifiable Secret Sharing scheme is used for creating fragments from the capabilities. If a peer is faulty or malicious, the verification computation allows the node to be efficiently detected without any network interaction. Since the fragments are stored in and retrieved or deleted from the Bamboo distributed hash table itself, no centralized reference monitor is needed.

3. GOALS

3.1 Resisting denials of service

Access control operations are on the critical path to reading or writing every protected object in the overlay. Preventing the execution of these operations can cripple a victim's data processing ability. There are three broad forms of attack: (a) saturate the network connections of the target servers with spurious data, (b) computationally overload the targets, and (c) exploit the node churn. The last attack is specific to peer-to-peer environments; if the distributed algorithm used relies on the continuous availability of all the participants, an attacker could exploit the fact that peer-to-peer overlays often have significant node churn. Otherwise, access control operations will be liable to denials of service when node churn occurs.

3.2 Detecting subverted nodes

In a distributed access control mechanism, nodes can attempt to disrupt the operation by either refusing to answer or providing a spurious response. The system must be able to tolerate such occurrences within predefined limits. In addition, it should be possible to determine which nodes are not operating correctly. An efficient approach is to design the protocol so that hostile nodes can be uniquely flagged using a suitable computational check on an auxiliary data field.

3.3 Preventing traffic analysis

Requests and responses in peer-to-peer overlays may pass through multiple nodes. A malicious node can monitor the traffic passing through it and build profiles of users that map them to subsets of their requests. In designing the access control layer, we must prevent it from revealing information that would allow such profiles to be constructed. In particular, a malicious node should not be able to determine the names of objects or the type of rights being requested by a remote user, even if the malicious node participates in the access control protocol. DAAL uses composed encryption to guard against metadata leaking.

3.4 Delinking storage from access control

The system responsible for storing data has traditionally also provided the necessary security guarantees. If an object was moved to a different system, these assurances were lost. Using encryption and hashing, it is possible to retain assurances about the confidentiality and integrity of data that crosses administrative domains. Mazieres argued for a global file namespace and modular mechanisms for managing the keys associated with the aforementioned operations [12]. Secure filesystems like Plutus [10] and Paranoid [21] do operate with a global namespace but require specific key management services. (Plutus uses one to provide lazy revocation of access keys while Paranoid's provides efficient group management using key transformations.) DAAL separates the storage operations from those needed for effecting access control.

4. MECHANISMS

4.1 Constructing permissions offline

Identity Based Encryption (IBE) [18] is a form of asymmetric cryptography that constrains the form of the public key. Specifically, the users choose their public key arbitrarily and then derive the corresponding private key. The utility of such a scheme lies in the fact that public keys would no longer need to be looked up online. If users identified each other using a canonical nomenclature, a recipient's name could serve as their public key. However, this glosses over one detail. The binding between an identity and the public key used is effected through the guarantee provided by a mutually trusted authority.

Since IBE does not use certificates, no online lookup needs to be performed. Instead, the process of generating the public and private key pair is dependent on information known only to the trusted authority. A user's knowledge of a decryption key corresponding to the user's identity is therefore equivalent to the user having been certified by the authority. Boneh and Franklin provided the first practical realization [3]. Their scheme uses a bilinear map $\hat{e}$ defined over two finite groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, of large prime order q. Here $\hat{e}(aP, bQ) = \hat{e}(P,Q)^{ab}$, where $P, Q \in \mathbb{G}_1$ and $\hat{e}(P,Q) \in \mathbb{G}_2$. $a, b \in \mathbb{Z}_q^*$. A hash function H_1 maps arbitrary binary strings to elements of $\mathbb{G}_1$. Another hash function H_2 maps elements of $\mathbb{G}_2$ to l bit strings. An administrator chooses a master key $s \in \mathbb{Z}_q^*$, selects a point $P \in \mathbb{G}_1$, and computes $P' = sP$. Everything except the master key is published. A user is given the private key $Q' = sQ$, where $Q = H_1(I)$ and I is the user's identity. The *Bilinear Diffie Hellman Assumption* posits that it is hard to compute s given Q and Q'.

Another user who sends a message, M, to user I, first selects a random number $r \in \mathbb{Z}_q^*$. The user encrypts the message by computing $C = M \oplus H_2(\hat{e}(Q, P')^r)$. C and rP are sent to the recipient. ($\oplus$ is bitwise exclusive-or.) The original message is recovered by computing $M = C \oplus H_2(\hat{e}(Q', rP))$. Since only the intended recipient has been given Q' by the administrator, only that recipient can perform the decryption. Transforming a user's identity, I, into that user's encryption key, Q, only requires computing $Q = H_1(I)$. No online lookup needs to be performed. DAAL utilizes this property to enable a peer user to construct an access right for a remote user in the absence of a public key infrastructure.

4.2 Identifying malicious participants

A user may wish to keep information such as decryption keys secret but may still wish to allow others to access the information under specific circumstances. One mechanism for this is to trust the secret to a group of size β. No individual or proper subset would be able to retrieve the secret. If the group's members all agree that the circumstances warrant it, then together they would be able to reconstruct the secret. Giving them portions of the secret would leak partial information about it. Instead the portions could be encrypted and the keys could be given to the group's members. Further, it may be preferable to allow a predefined threshold, α, of the group's membership to decide to reconstruct the secret. An example of the need for this is the situation when some members are absent. Such a scheme can be implemented by recursively encrypting keys and sharing the results with appropriate subsets of the group. However, the number of encryption operations needed and keys used grows exponentially.

Secret sharing using interpolation [17] provides an efficient solution. The secret is represented with an $(\alpha - 1)$ degree polynomial. Each of the group's members is given the result of evaluating the polynomial at one of β points. Since any α points fully specify the polynomial, they are enough to reveal the secret. Fewer points reveal no information. DAAL leverages this property to distribute an access control permission over β peers while needing the cooperation of only α of them for requests to succeed, as depicted in Figure 1.

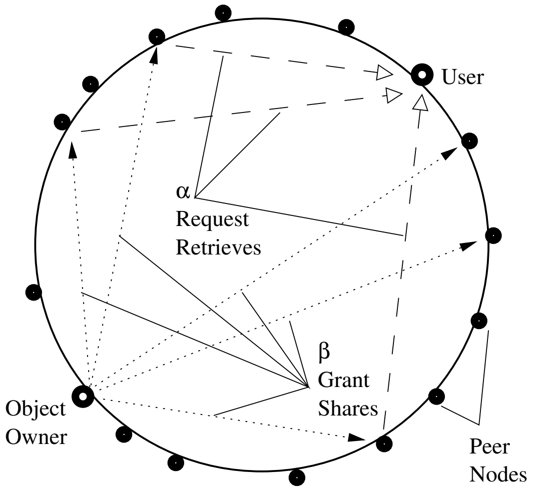

Figure 1: Granting a permission results in β shares being inserted in the overlay. Requesting a permission requires at least α shares to be retrieved.

The owner of the secret may provide a faulty share to a member of the group. If this occurs, the others may accuse the member of attempting to subvert the reconstruction protocol. To guard against this, *verifiable secret sharing* can be used [13]. In this scheme, each share is accompanied by an auxiliary piece of information. The member can use the latter to verify the legitimacy of the share. DAAL uses this functionality differently. The model assumes that the object owner will not provide faulty shares. Instead it expects that some of the peers may act maliciously. When a user attempts to reconstruct a permission, there may be a problem with one or more of the shares that it retrieves. If this occurs, the scheme's non-interactive verification property allows DAAL to efficiently detect the set of subverted nodes without any further network communication.

DAAL uses Pedersen's scheme as described here. A prime $p = mq + 1$ is chosen, where m is a small integer and q is a large prime. Next, $g \in \mathbb{Z}_p^*$ and $d \in \mathbb{Z}_q^*$ are randomly selected. g must have order q, that is $g^q = 1 \bmod p$. $h = g^d \bmod p$ is calculated. Assume the secret $x \in \mathbb{Z}_q^*$ is to be split into β shares. Two $(\alpha - 1)$ degree polynomials, γ and δ, are chosen with random coefficients in $\mathbb{Z}_q^*$ and subject to the constraint that $\gamma(0) = x$. γ_m and δ_m denote the coefficients of the m^{th} order term of γ and δ, respectively. $\epsilon_m = g^{\gamma_m} h^{\delta_m}$ is calculated for each $m \in \{0, \ldots, (\alpha - 1)\}$. All ϵ_m are provided to each member. The i^{th} share of x is $\{\gamma(i), \delta(i)\}$. These two elements are the polynomials γ and δ evaluated at i, respectively. A share's consistency can be verified as follows. p, g, h are global parameters. The i^{th} recipient receives $\{\gamma(i), \delta(i)\}$ and all ϵ_m and thus can check if $g^{\gamma(i)} h^{\delta(i)} \overset{?}{=} (\epsilon_0)(\epsilon_1)^i (\epsilon_2)^{i^2} \cdots (\epsilon_{\alpha-1})^{i^{\alpha-1}} \pmod p$. DAAL uses this property to verify that the i^{th} peer is providing legitimate responses. If the check fails, the peer is deemed to be malicious.

4.3 Reliable dispersal of permission fragments

In principle, DAAL can store its access control metadata in any peer-to-peer system. In practice, the protocol uses a *distributed hash table* (DHT) [15] for three reasons. The first is the search mechanism. Unstructured overlays allow the use of complex query resolution procedures. When a query is performed, an exact match may not be found. Instead one that is deemed close enough may be returned. DAAL object names and contents are hashed and encrypted. As a result, no metric of closeness can be used. Further, the overlay will yield only false matches when a node storing a DAAL object is offline. Therefore, additional functionality in DAAL would be needed. It would have to disambiguate legitimate responses from those returned in lieu of the specific requested object.

The second reason is the interface. DHTs provide operations to put, get, and remove objects. This is the minimal functionality necessary for DAAL to store its capability shares. When an object is inserted, an auxiliary credential is provided. Removes succeed only if this credential is provided. This serves as peer-to-peer level authentication. DAAL leverages this so that only the user who inserted an access right into the system can remove it. If this functionality was not provided, DAAL would need to add it atop the overlay. Without it, DAAL would not be able to restrict revocation operations to the legitimate owners of objects. The Bamboo [15] operations used by DAAL are $put(address, key, value, secret)$, $get(address, key)$, and $remove(address, key, secret)$. $address$ is the peer acting as the gateway for the operation. The object $value$ is stored under name key. The $secret$ provided when an object was inserted is required for a removal to succeed.

The third reason is performance. Assume a peer-to-peer system has n nodes. The expected response time in an unstructured peer-to-peer overlay is $O(n)$. In a structured overlay, it is $O(\log n)$. DAAL can operate correctly with the slower query process. However, DAAL operations are on the critical path to gaining access and committing changes to an object. Hence their latency is noticeable. Using a DHT avoids this performance penalty.

5. PROTOCOL DESIGN

DAAL consists of seven operations. The first two are used offline by an administrator to initialize global parameters and create new user identities. An object's owner can use the next two operations to grant and revoke permissions for it. The last three operations allow any authorized user to retrieve access rights, transform data into protected DAAL objects, and read such objects transparently.

5.1 Bootstrap

In Section 4.1, the bilinear mapping $\hat{e}$ and groups $\mathbb{G}_1, \mathbb{G}_2$ were left unspecified. The DAAL prototype uses the Tate pairing, τ, [5]. An elliptic curve $E(\mathbb{F}_{p'})$ defined over a finite field $\mathbb{F}_{p'}$ of prime order p' serves as $\mathbb{G}_1$, while $\mathbb{G}_2 = \mathbb{F}^2_{p'}$. p' exceeds 1024 bits. During initialization the administrator, A, creates the master key s and other parameters described in Section 4.1.

$$\begin{array}{lll} A & : & s \\ A & : & \xi_{IBE} \longleftarrow \{\ \hat{e} = \tau,\ \ \mathbb{G}_1 = E(\mathbb{F}_{p'}),\ \ \mathbb{G}_2 = \mathbb{F}^2_{p'}, \\ & & \quad q,\ \ l,\ \ H_1,\ \ H_2,\ \ P,\ \ P' = sP\ \} \end{array}$$

Next, the administrator creates the global parameters described in Section 4.2. Default values for α and β are also selected. A user is free to change α and β on a per object basis. For example, the user may wish to decrease $(\beta - \alpha)$ to increase the likelihood of permission revocation succeeding for a particular data object. Adjusting these alters the security assurances only for that user's data. It does not affect data owned by other users.

$$A \ :\ \xi_{VSS} \longleftarrow \{\ p,\ \ g,\ \ h,\ \ \alpha,\ \ \beta\ \}$$

The administrator then selects a set of nodes, $\{a_i\}, i \in \{1, 2, \ldots\}$, in the DHT overlay. These allow a DAAL client to bootstrap the connection to the overlay if the user's node is not already a member.

$$A \ :\ \xi_{DHT} \longleftarrow \{\ a_i\ \}$$

A hashing algorithm, H_3, digital signature scheme, S^{-1}, and symmetric cipher, S, have to be used to ensure the confidentiality and integrity of the objects protected by DAAL. In principle, these algorithms could be different for each object. In practice, this is unlikely to yield any benefit because the same standardized algorithms, like DSA for signing, SHA-256 for hashing and AES for encryption, are likely to be used for all objects. Additionally, it would increase the metadata associated with each object. Instead, the administrator selects them once for the entire system.

$$A \ :\ \xi_{HSE} \longleftarrow \{\ H_3,\ \ S^{-1},\ \ S\ \}$$

Finally, the set of global parameters, ξ, is created by combining the above.

$$A \ :\ \xi \ \longleftarrow\ \ \xi_{IBE}\ \cup\ \xi_{VSS}\ \cup\ \xi_{DHT}\ \cup\ \xi_{HSE}$$

5.2 Create user

A subject's identity, I, must be certified by an authority trusted by all the members of a peer group. The process can occur either offline or by using an out-of-band channel. An example of the latter would be the use of secure email. This step must be completed before the subject can use DAAL to grant or request access to an object. The subject provides the administrator with suitable credentials identifying the subject. The administrator constructs a private key, Q', and public key, Q, corresponding to the subject's identity.

$$\begin{array}{l} I \ \longrightarrow_{offline}\ A\ :\ I \\ A\ :\ \zeta_E \ \longleftarrow \{\ Q' = sQ,\ \ Q = H_1(I)\ \} \end{array}$$

A user who reads an object first verifies its integrity. This is done by checking that the object was modified with a valid write capability. An attacker that modifies the object could also replace its integrity verification key. To detect this, a legitimate integrity verification key must be signed by the owner. Each subject therefore needs a signing key, I_s, for use with the digital signature algorithm, S^{-1} specified in ξ. Further, any user must be able to verify the signature without resorting to a centralized online lookup. As a result, I_s's corresponding signature verification key, I_v, must be included in the object. To prevent an attacker from substituting I_v with a false one, it must be certified by the administrator. Thus a certificate, $I_c = sH_1(I \| I_v)$, is also included. ($\|$ denotes concatenation.) s is the administrator's master key as described in Section 4.1. Only the administrator can generate such an I_c, effectively making it a certificate binding I to I_v. Its use is explained in Section 5.4.

$$A\ :\ \zeta_S \ \longleftarrow \{\ I_s,\ \ I_v,\ \ I_c\ \}$$

Finally, the administrator provides the subject with credentials and the set of global parameters, ξ, needed to implement DAAL access control operations.

$$I \ \longleftarrow_{offline}\ A\ :\ \zeta_E\ \cup\ \zeta_S\ \cup\ \xi$$

5.3 Write object

No DAAL metadata exists for an object before it is written out for the first time. The owner must therefore create a read capability, κ_r, and a write capability, κ_w for the object. κ_r is a key for the symmetric cipher, S, specified in ξ. κ_w is the signing key for the digital signature algorithm, S^{-1}, specified in ξ. Subsequently, these are retrieved as described in Section 5.7. When κ_w is generated, an accompanying verification key, κ_v, is created. It will allow the validity of the signed hashes to be checked.

$$I\ :\ \kappa_r,\ \ \kappa_w,\ \ \kappa_v$$

Unlike κ_r and κ_w, κ_v is stored in the protected DAAL object itself. Only the owner should be able to create a legitimate κ_v. It must also be bound to a specific object, O. In the absence of these two properties, an unauthorized user could modify the object and replace the signed hash without being detected. Hence the object's name and integrity verification key are signed by the owner, yielding $V'(O) = S^{-1}(I_s, O \| \kappa_v)$. The set $V(O) = \{\ I_c, I_v, V'(O), \kappa_v\ \}$ establishes a certification chain from the administrator, through the object's owner, to the signed hash attached to the object. Therefore, $V(O)$ serves as a certified verification key for the object. This process needs to be done only the first time the object is written out and when the write capability is changed after a user's write permission is revoked. Thereafter, the same $V(O)$ is retained and used to verify the object's integrity when it is read.

$$\begin{array}{lll} I &:& V'(O) \longleftarrow S^{-1}(I_s, O \| \kappa_v) \\ I &:& V(O) \longleftarrow \{\ I_c,\ I_v,\ V'(O),\ \kappa_v\ \} \end{array}$$

During subsequent writes, the following occurs. The contents of the object named O are denoted by $M(O)$. The hash, H_3, specified in ξ is used to calculate a checksum $H(O) = H_3(M(O))$. It is then signed to produce $C^{-1}(O) = S^{-1}(\kappa_w, H(O))$. $C^{-1}(O)$'s dependence on the write capability prevents unauthorized users from generating a verifiable checksum. The object is then encrypted with the cipher S and key κ_r yielding an encrypted version, $C(O) = S_e(\kappa_r, M(O))$. S_e denotes S used in encryption mode. Only users with the read capability will be able to decrypt it and check its integrity. A protected DAAL object, $D(O)$, is constructed as shown below. The user, I, places it in storage at node N, which may be a peer in the overlay, a remote server, or just local storage.

$$\begin{array}{lll} I &:& H(O) \longleftarrow H_3(M(O)) \\ I &:& C^{-1}(O) \longleftarrow S^{-1}(\kappa_w, H(O)) \\ I &:& C(O) \longleftarrow S_e(\kappa_r, M(O)) \\ I &:& D(O) \longleftarrow \{O, I, \alpha, \beta, V(O), C^{-1}(O), C(O)\} \\ I \longrightarrow N &:& D(O) \end{array}$$

5.4 Read object

To read an object O, the user must have the requisite permission, κ_r. Section 5.7 describes how access rights are obtained. The user, J, retrieves the DAAL object, $D(O)$, from node N. $D(O)$ is split into the constituent pieces, described in Section 5.3. The cipher, S, specified in ξ and read capability, κ_r, are used to decrypt $C(O)$ obtained from $D(O)$. This produces the decrypted object O with contents $M(O) = S_d(\kappa_r, C(O))$. S_d denotes S used in decryption mode.

$$\begin{array}{lll} N \longrightarrow J &:& D(O) \\ J &:& M(O) \longleftarrow S_d(\kappa_r, C(O)) \end{array}$$

The following process is used to check that the object has been written with authorization. The contents of the object are hashed to $H(O) = H_3(M(O))$ using H_3 specified in ξ. This is verified by evaluating $S_v^{-1}(\kappa_v, C^{-1}(O), H(O))$. S_v^{-1} is the digital signature algorithm, S^{-1}, specified in ξ, used in signature verification mode. κ_v is obtained from $V(O)$. $V(O)$ and $C^{-1}(O)$ are extracted from $D(O)$. If the signature does not match, the object has been modified by a user without write permission.

$$\begin{array}{lll} J &:& H(O) \longleftarrow H_3(\ M(O)\ ) \\ J &:& true \mid false \longleftarrow S_v^{-1}(\kappa_v, C^{-1}(O), H(O)) \end{array}$$

If the signature matches, the ownership of the integrity verification key, κ_v, must be ascertained. This prevents a user other than the owner from replacing κ_v. An attacker who could replace κ_v would be able to generate a valid signature, $C^{-1}(O)$. $S_v^{-1}(I_v, V'(O), O \| \kappa_v)$ is evaluated using I_v and $V'(O)$ from $V(O)$. If the signature is false, the integrity verification key has been substituted without authorization.

$$J \ :\ true \mid false \longleftarrow S_v^{-1}(I_v, V'(O), O \| \kappa_v)$$

If it matches, the final step in the verification must be performed. I_c from $V(O)$ is used to check that I_v was issued by the administrator. A random number $r \in \mathbb{Z}_q^*$ is selected. If I_v is a legitimate signing key for owner I, then $\hat{e}(H_1(I \| I_v), P')^r \stackrel{?}{=} \hat{e}(I_c, rP)$ should hold. The object's owner, I, is obtained from $D(O)$. $q, \hat{e}, P, P', H_1$ are taken from ξ. Once this check succeeds, the object's integrity is completely verified.

$$J \ :\ \hat{e}(H_1(I \| I_v), P')^r \stackrel{?}{=} \hat{e}(I_c, rP)$$

5.5 Grant permission

The owner, I, of an object, O, can give another user, J, the permission to read or write it. Possession of the read capability, κ_r, or write capability, κ_w, for the object is equivalent to having read or write permission, respectively. The task of granting a permission is therefore transformed into the problem of transferring the appropriate capability from I to J. A capability set θ is constructed as $\theta = \kappa_r \| \emptyset$ for read permission, $\theta = \emptyset \| \kappa_w$ for write permission, or $\theta = \kappa_r \| \kappa_w$ for both permissions. $\emptyset$ denotes an empty field. DAAL implements only read and write permissions. Generalizing this to a larger set of permissions requires only the definition of θ to be changed.

$$I \ :\ \theta \longleftarrow \begin{cases} \kappa_r \| \emptyset \\ \emptyset \| \kappa_w \\ \kappa_r \| \kappa_w \end{cases}$$

The capability set is encrypted using J's identity. Consequently, only J can recover the permission. The encryption is done by choosing a random number $r \in \mathbb{Z}_q^*$ and computing $\theta' = \theta \oplus H_2(\hat{e}(H_1(J), P')^r)$ where $q, P, \hat{e}, H_1, H_2$ are from ξ and described in Section 4.1. The encrypted capability set is $\lambda = \{rP, \theta'\}$.

$$\begin{array}{lll} I &:& \theta' \longleftarrow \theta \oplus H_2(\ \hat{e}(H_1(J), P')^r\ ) \\ I &:& \lambda \longleftarrow \{rP, \theta'\} \end{array}$$

λ is split into β shares using Pedersen's scheme. Two $(\alpha - 1)$ degree polynomials, γ and δ, are chosen with random coefficients subject to the constraint $\gamma(0) = \lambda$. ϵ_m are calculated as described in Section 4.2. p, g, and h are obtained from ξ. ($\in_R$ denotes random selection.)

$$\begin{array}{lll} I &:& \gamma(0) = \lambda, \quad \gamma_m, \delta_m \in_R \mathbb{Z}_q^* \\ I &:& \underset{m \in \{0,\dots,\alpha-1\}}{\forall m} \quad \epsilon_m \longleftarrow g^{\gamma_m} h^{\delta_m} \end{array}$$

γ and δ are evaluated at β points. The two values obtained at the n^{th} point are combined with the set of α values in $\{\epsilon_m\}$ to produce π_n, the n^{th} share of the capability set.

$$I \ :\ \underset{n \in \{1,\dots,\beta\}}{\forall n} \quad \pi_n \longleftarrow \gamma(n),\ \ \delta(n),\ \ \{\epsilon_m\}$$

Each share must be stored at a different peer. $H_3(J \| O \| n)$ is used as the DHT key for the n^{th} share. Since H_3 is cryptographically strong, the shares will be distributed to random points in the overlay. To prevent a single gateway from subverting the protocol, a random peer a_n is chosen from the set of addresses provided in ξ. The *put* operation is effected at a_n. Each user maintains a single secret key, κ_d. This is used as an authenticator for all its DHT *put* and *remove* operations. It is combined with the share details to yield a unique key $\kappa'_d = H_3(J\|O\|n\|\kappa_d)$.

$$\underset{n \in \{1,\ldots,\beta\}}{\forall n} \left\{ \begin{array}{lll} I & : & a_n \in_R \{a_i\} \\ I & : & \kappa'_d = H_3(J\|O\|n\|\kappa_d) \\ I & \rightarrow & a_n : put(a_n, H_3(J\|O\|n), \pi_n, \kappa'_d) \end{array} \right.$$

5.6 Revoke permission

The owner, I, of an object, O, can revoke user J's permissions for it. To do this, the owner attempts to remove all β shares of the capability set from the overlay. As long as $(\beta-\alpha+1)$ or more are removed, the revocation succeeds. $H_3, \{a_i\}$ are from ξ. κ_d, κ'_d are as described in Section 5.5.

$$\underset{n \in \{1,\ldots,\beta\}}{\forall n} \left\{ \begin{array}{lll} I & : & a_n \in_R \{a_i\} \\ I & : & \kappa'_d = H_3(J\|O\|n\|\kappa_d) \\ I & \rightarrow & a_n : remove(a_n, H_3(J\|O\|n), \kappa'_d) \end{array} \right.$$

5.7 Request permission

A user, J, can request permission to read or write an object, O, as follows. The user first attempts to retrieve the β shares of the object's encrypted capability set from the overlay. The value of β is the one specified in the DAAL protected version of the object. $H_3, \{a_i\}$ are from ξ.

$$\underset{n \in \{1,\ldots,\beta\}}{\forall n} \left\{ \begin{array}{l} J \ : \ a_n \in_R \{a_i\} \\ J \leftarrow a_n : \pi_n \leftarrow get(a_n, H_3(J\|O\|n)) \end{array} \right.$$

Using Pedersen's scheme, described in Section 4.2, the user verifies each share. p, g, h are from ξ. $\gamma(n)$, $\delta(n)$, $\{\epsilon_m\}$ are from π_n. If the check on π_n fails, the peer a_n is flagged as malicious. The check is done modulo p.

$$\underset{n \in \{1,\ldots,\beta\}}{\forall n} \quad g^{\gamma(n)} h^{\delta(n)} \stackrel{?}{=} (\epsilon_0)(\epsilon_1)^n(\epsilon_2)^{n^2} \cdots (\epsilon_{\alpha-1})^{n^{\alpha-1}}$$

If at least α shares can be verified, γ can be reconstructed by interpolating α values of it. By evaluating it at 0, the encrypted capability set can be extracted.

$$\begin{array}{lll} J & : & \gamma \longleftarrow \{\gamma_n\} \\ J & : & \lambda \longleftarrow \gamma(0) \end{array}$$

A user with the IBE private key J', and rP, θ' from λ, can calculate the capability set θ. $\hat{e}, H_2$ are from ξ. θ contains κ_r, κ_w or both according to the permissions the owner granted J.

$$J \ : \ \theta \ \longleftarrow \ \theta' \oplus H_2(\ \hat{e}(J', rP)\ )$$

6. IMPLEMENTATION

6.1 Cautionary notes

Security protocols must be carefully designed. Subtle changes, such as the ordering of operations, may result in significant weaknesses. Therefore, we solicited feedback about implementing DAAL's protocols. This allowed us to identify three errors that might arise during implementation. They motivate the need for a simple programming interface that reduces the role of the application developer in the process of utilizing DAAL. We first describe the errors and then outline how we abstracted each of DAAL's steps behind a minimal Java programming interface.

Two potential implementation errors were identified in the process of granting a permission. In the first case, a developer may opt to use a single gateway when inserting permission fragments into the overlay, reasoning that the fragments are destined to reside at a range of nodes but ignoring the fact that the gateway can form a critical point of failure. In practice, such an implementation would give a gateway the power to completely deny all access control operations performed by clients connecting through it.

The second case will not affect protocol correctness but can leave the system exposed to a storage-exhaustion denial-of-service attack. It may be reasoned that it is more computationally efficient to select a set of random DHT keys for the permission fragments rather than constructing them as specified (as $H_3(J||O||n)$). To allow the permission's recipient to know where to recover the permission fragments from, the random keys are embedded in the object's metadata. As the number of users, objects and permission shares increase, the number of keys grows multiplicatively, with the result that the storage needed for the metadata will dominate that required for the data itself.

The third error is more subtle. An alteration to the protocol may be made that exposes the read key whenever the write key is granted. This may (falsely) be viewed as safe because the read key is always needed to verify the integrity of the object. However, the semantics of write operations only specify that the new object's integrity must be verifiable. No assurance is to be provided about the integrity of the previous version of the object. This is analogous to the semantics of local storage where the old version of a file is not verified, but simply overwritten. The modified protocol would introduce a breach of confidentiality for all objects for which write but not read permission is granted.

6.2 Operation interfaces

We now describe how each operation's protocol is abstracted behind a simple programming interface to facilitate correct usage.

6.2.1 Bootstrapping

The bootstrap protocol steps described in Section 5.1 are implemented in DAAL's **Globals** and **Administrator** classes. An administrator initializes the system by invoking constructors as shown below. The second step results in the `administrator.masterKey` needed for creating new users.

```
globals = new Globals();
administrator = new Administrator(globals);
```

6.2.2 Creating a user

An administrator can create a user with identity `I` by invoking the constructor of DAAL's **User** class with the `masterKey` and `globals` parameters from Section 5.1, as follows. This implements the steps described in Section 5.2.

```
user = new User(masterKey,globals,"I");
```

Since the **User** and **Globals** classes are serializable, the `user` and `globals` instances can be passed to the new user as files or through Java RMI. The user then constructs a `delegation` instance of DAAL's **Delegation** class to hold the permissions the user delegates to other users.

```
delegation = new Delegation();
```

6.2.3 Writing an object

A user can write an object by invoking the **seal()** method in DAAL's **Data** class, which implements the steps described in Section 5.3. A file named `O` can be protected (so that it can be safely exported to an untrusted remote node in the overlay) with the parameters from Section 5.2 as follows:

```
Data.seal(globals,delegation,user,"O",alpha, beta);
```

Smaller values of α yield faster access requests, while lower values of $(\beta - \alpha)$ yield faster revocation. Simultaneously, larger values of $\frac{\alpha}{\beta}$ improve resilience to malicious overlay node activity. Thus, `alpha` and `beta` can be selected on the basis of the application that will use the data.

6.2.4 Reading an object

A user can read an object by invoking the **unseal()** method in DAAL's **Data** class, which implements the steps described in Section 5.4. A file `O` retrieved from the overlay can be read or written with suitable `capabilities` (that can be requested by following the steps in Section 5.7):

```
Data.unseal(globals,"O",capabilities);
```

6.2.5 Granting permission

An object's owner can grant permission to a user by invoking the **grant()** method in DAAL's **Access** class, which implements the steps described in Section 5.5. An object's owner passes `owner`, its instance of DAAL's **User** class, along with `J`, the identity of the delegatee, and the file name `O` to the **grant()** method. If the owner wishes to grant read permission, `grantRead` should be `true`. Similarly, `grantWrite` should be set to `true` for write permission.

```
Access.grant(globals,delegation,owner,"J","O",
                             grantRead,grantWrite);
```

6.2.6 Revoking permission

The access rights for file `O`, granted by `owner` to user with identity `J`, can be retracted by calling the **revoke()** method of DAAL's **Access** class, as shown below. This implements the steps described in Section 5.6

```
Access.revoke(globals,delegation,owner,"J","O");
```

6.2.7 Requesting permission

A user can request access rights for a file `O`, by passing its instance of DAAL's **User** class and the filename to the **request()** method of DAAL's **Access** class. The steps described in Section 5.7 are implemented by the method that then returns `capabilities`, an instance of DAAL's **Capabilities** class that contains the object's decryption and signing keys (that proxy for read and write permissions, respectively) if they have been granted by the object's owner.

```
capabilities = Access.request(globals,user,"O");
```

6.3 Using DAAL over PlanetLab

To gain insight into DAAL's utility, we deployed it over PlanetLab, which consists of several hundred nodes in over thirty-five countries. Table 1 presents the time it takes to complete each of DAAL's operations. The prototype is a combination of the Java Cryptography Architecture, an implementation of Boneh-Franklin identity-based encryption using the modified Tate pairing [4], Pedersen's verifiable secret sharing [13], and the Bamboo [15] DHT running on Planetlab. The underlying platform of each DAAL peer was Mac OS 10.4.3 on a 1.2 GHz PowerPC. The remaining peers are varied but conform to the Planetlab node specification. Access control for all operations used parameters of $\alpha = 3$ and $\beta = 4$. SHA was used for hashing, 1024-bit DSA for signing, and 40-bit RC4 for symmetric encryption of data. DHT operations were performed with a 60-second time-to-live. The data file was 1.5 KB of text.

Each operation takes multiple seconds to complete. However, profiling the prototype reveals that the reasons for this are diverse. In the case of bootstrapping and creating a user, most of the time is taken for key generation. The time to perform DHT operations is the dominant cost for requesting and revoking a permission. Further, DHT operations contribute a large fraction of the cost for granting a permission. The other large cost for granting a permission and the primary cost for reading an object is the time spent computing the bilinear mapping over the elliptic curve. Writing an object completes rapidly as it uses traditional cryptographic primitives that have been optimized for performance. The code used to perform the bilinear mapping is written in Java. An alternative C implementation of the Boneh-Franklin primitives is faster. It uses the Weil pairing instead of the Tate pairing, and the elliptic curves are defined over a different polynomial. Permission grants and object reads could be sped up through the use of this code.

Operation	Time (s)
Bootstrap	8.5
Create user	12.7
Read object	64.2
Write object	0.2
Grant permission	51.9
Revoke permission	14.9
Request permission	10.3

Table 1: Time for each DAAL operation.

7. RELATED WORK

Traditional distributed storage systems [20, 9] had a different trust model from current peer-to-peer networks. While the users were not trusted, the client hosts from which they connected were assumed to be running system software that was controlled by administrators. As the size of the deployments increased, ensuring the integrity of every machine's software became more difficult. Athena and Andrew [16] addressed this by redefining the trusted computing base to exclude all software running on workstations. Kerberos's [19] authentication and authorization services ran on a few machines where users could not log in. CRISIS [1] was designed to provide the same services across the wide area where network connectivity is unreliable. However, their dependence on specialized security nodes limits their utility in peer-to-peer environments for the reasons described in Section 3.

A number of projects [10, 21] adapted the idea of allowing clients to store data on untrusted remote servers. Access control is provided using cryptographic primitives. These systems require specialized group servers that cannot be run on an untrusted node. Also, each server manages its own authentication. SFS [12] was extended with a decentralized authentication server, which allows cryptographic credentials to be transparently exchanged between SFS file servers in different protection domains. Its reliance on these specialized servers rules out its use in the peer-to-peer context. SiRiUS [7] was designed for use over a range of storage technologies, including peer-to-peer overlays. However, it does not address the problem of continued operation in the face of malicious nodes. In contrast, DAAL's use of verifiable secret sharing for the authorization metadata allows it to tolerate a fraction of the overlay operating maliciously.

8. CONCLUSION

We described our implementation of DAAL, an access control mechanism for peer-to-peer overlay networks. DAAL does not require an online authentication server nor does it utilize a centralized reference monitor. Subjects' identities are cryptographically constructed so that they can be used by remote nodes without needing to do any network queries. Objects are transformed cryptographically so that they can be accessed and legitimately modified only with suitable rights that are defined in the form of cryptographic capabilities. The rights are processed using a cryptographic protocol so that they can be spread over a significant number of nodes in the overlay. Using a right requires it to be reconstituted from any subset (of a configurable size) of the aforementioned nodes. Revocation is effected by contacting at least a threshold number of those nodes. In effect, the overlay's storage functionality is coupled with cryptographic protocols and utilized to provide the functionality of a reference monitor.

9. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grants OCI-0722068 and CNS-0447671. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

10. REFERENCES

[1] E. Belani, A. Vahdat, T. Anderson, and M. Dahlin, CRISIS Wide Area Security Architecture, 7th USENIX Security Symposium, 1998.

[2] Steven M. Bellovin and Michael Merritt, Limitations of the Kerberos Authentication System, USENIX Conference, 1991.

[3] Dan Boneh and Matt Franklin, Identity-based Encryption from the Weil Pairing, SIAM Journal of Computing, 32(3), 2003.

[4] A. Duffy and T. Dowling, An Object Oriented Approach to an Identity Based Encryption Cryptosystem, 8th IASTED International Conference on Software, 2004.

[5] Steven Galbraith, Keith Harrison and David Soldera, Implementing the Tate Pairing, Algorithmic Number Theory Symposium, Lecture Notes in Computer Science 2369, 2002.

[6] Ashish Gehani and Surendar Chandra, Parameterizing Access Control for Heterogeneous Peer-to-Peer Applications, 3rd International Conference on Security and Privacy in Communication Networks (SecureComm), IEEE Computer Society, 2007.

[7] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu and Dan Boneh, SiRiUS: Securing Remote Untrusted Storage, Network and Distributed Systems Security Symposium, 2003.

[8] Michael A. Harrison, Walter L. Ruzzo and Jeffrey D. Ullman, Protection in Operating Systems, CACM, 19(8), 1976.

[9] A. Hisgen, A. Birrell, T. Mann, M. Schroeder and G. Swart, Availability and Consistency Tradeoffs in the Echo Distributed File System, 2nd IEEE Workshop on Workstation Operating Systems, 1989.

[10] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang and Kevin Fu, Plutus: Scalable Secure File Sharing on Untrusted Storage, 2nd Conference on File and Storage Technologies, 2003.

[11] Butler W. Lampson, Protection, 5th Princeton Symposium on Information Sciences and Systems, 1971.

[12] David Mazieres, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel, Separating Key Management from File System Security, 17th ACM Symposium on Operating Systems Principles, 1999.

[13] T. P. Pedersen, Non-interactive and Information-theoretic Secure Verifiable Secret Sharing, Advances in Cryptology, Lecture Notes in Computer Science 576, 1991.

[14] http://www.planet-lab.org

[15] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica and Harlan Yu, OpenDHT: A Public DHT Service and Its Uses, ACM SIGCOMM, 2005.

[16] Mahadev Satyanarayanan, Integrating Security in a Large Distributed System, ACM Transactions on Computer Systems, 7(3), 1989.

[17] A. Shamir, How to Share a Secret, CACM, 22(11), 1979.

[18] A. Shamir, Identity-based Cryptosystems and Signature Schemes, Advances in Cryptology, Lecture Notes in Computer Science 196, 1984.

[19] J. G. Steiner, B. C. Neuman and J. I. Schiller, Kerberos: An Authentication Service for Open Network Systems, Winter Usenix Conference,1988.

[20] Edward Wobber, Martin Abadi, Michael Burrows, and Butler Lampson, Authentication in the Taos Operating System, 14th ACM Symposium on Operating System Principles, 1994.

[21] Fareed Zaffar, Gershon Kedem and Ashish Gehani, Paranoid: A Global Secure File Access Control System, 21st Annual Computer Security Applications Conference, 2005.