11-9-2008

IDEAS: An Identity-based Security Architecture for Large-scale and High-performance Storage Systems

Zhongying Niu
Wuhan National Laboratory for Optoelectronics, China

Ke Zhou
Wuhan National Laboratory for Optoelectronics, China

Hong Jiang
University of Nebraska-Lincoln, jiang@cse.unl.edu

Dan Feng
Wuhan National Laboratory for Optoelectronics, Wuhan, 430074, China

Tianming Yang
Wuhan National Laboratory for Optoelectronics, China

University of Nebraska–Lincoln, Computer Science and Engineering
Technical Report TR-UNL-CSE-2008-0013
Issued Nov. 9, 2008

IDEAS: An Identity-based Security Architecture for Large-scale and High-performance Storage Systems

Zhongying Niu[†], Ke Zhou[†], Hong Jiang[‡], Dan Feng[†], Tianming Yang[†]
[†]*College of Computer Science and Technology*
Huazhong University of Science and Technology
Wuhan National Laboratory for Optoelectronics, Wuhan, 430074, China
Email: niel@smail.hust.edu.cn, {k.zhou, dfeng}@hust.edu.cn
{tmyang}@smail.hust.edu.cn
[‡]*Department of Computer Science and Engineering*
University of Nebraska-Lincoln, Lincoln, NE 68588, USA
Email: jiang@cse.unl.edu

Abstract

We develop IDEAS, an identity-based security architecture for large-scale and high-performance storage systems, designed to improve security, convenience and total cost of access control by merging identity management with access control in these systems. IDEAS authenticates users at each I/O node by using a single-identity certificate without the service of a centralized security server and enforces access control mechanism by using an object-based access control (OBAC) model, which is designed to address the complexity and scalability issue of security administration in large-scale storage systems. We also discuss the issue of how to identify and authenticate a large number of users with the state-of-the-art cryptographic solutions and suggest the potential alternative technologies to the well-known PKI mechanism. In particular, we present a generic definition and formal description of the OBAC model. The access control rules for OBAC, namely, the PIPS (Proximity, Inheritance, Priority, Sharing) rules, proposed in this paper can be used as the basis for establishing a testing and evaluation criteria for securing general large-scale storage systems.

Experiments on the IDEAS prototype in the HUST OSD project show that IDEAS significantly outperforms the conventional capability-based security scheme (CapSec) in terms of latency for key security-related operations, by a speedup factor of 1.81 and 2.22 for the frequent read and write operations respectively and by a factor of 1.65, 1.22, and 0.52 for the infrequent create, delete and chmod operations respectively. Furthermore, in addition to achieving higher security, IDEAS drastically improves scalability by completely removing the performance bottleneck caused by security overhead through avoiding capability requests for both read and write operations, as evidenced by the zero read and write latency of IDEAS on the metedata server while CapSec quickly saturates its metadata server with a moderate number of read or write requests.

1. Introduction

Today, scientific computing, engineering design and simulation, as well as large commercial application have raised demands for large-scale and high-performance storage systems. High-performance computing (HPC) applications of today and tomorrow, such as High-Energy Physics, Biosciences, Chemistry, Astrophysics, and Geophysics call for high-performance data access and terabytes to petabytes of data storage. Large commercial applications such as Google and Yahoo may service millions of users and produce enormous amount of electronic data. Large-scale storage systems in the near future will require much larger data storage and higher data throughput. A typical large-scale storage system or next-generation storage system may service millions of clients and hold hundreds of metadata servers (MDS) and hundreds of thousands of storage devices, with petabytes to exabytes of storage capacities and hundreds of terabytes of aggregate bandwidth. Securing such a large-scale and high-performance storage system presents new challenges because of the large number of clients and concurrent accesses of both random I/O and high data throughput. The primary challenges that we address in this paper are listed below.

Challenge 1: *Added threat environment*. Recent studies [1]–[7] on large-scale and high-performance storage systems have enabled direct interaction between clients and storage devices. The storage devices are attached to the client-network, which enables the clients to directly access data from the storage devices to improve the

performance and scalability of the system. However, as storage systems and individual storage devices themselves become networked, they must defend against the attacks not only on the stored data itself but also on the messages traversing an untrusted public network.

Challenge 2: *Rapid authorization.* Due to the large number of nodes, the huge size of data sets, and the concurrency of their accesses, high performance computing and data-intensive applications generate an extremely high aggregate I/O demand on the storage subsystem. File accesses and I/O requests are often both extremely bursty and highly parallel [8] in high-performance storage systems. The efficiency of rapidly authorizing I/O requests directly affects the overall performance of the system.

Challenge 3: *Complex security management.* The main task of security administration is to maintain user's identity and access privilege information. Commonly, the identity information is stored in a local user database and the access privilege information is organized in the form of access control list or matrix. Given the scale of the systems under consideration, the user database and access control list or matrix can become too large and complex to maintain and operate efficiently and economically.

Challenge 4: *Identifying and authenticating an enormous number of clients.* The world is becoming identity based (e.g., universal and unique IDs being implemented in several countries for their citizens). Large-scale storage systems may service millions of clients from different organizations and with multiple distinctive identities each. Conventional identification and authentication technologies provided by ad-hoc, single-purpose systems, such as global user lists and password based systems are not sufficiently competent to identify and authenticate such an enormous number of clients. In addition, identification technologies and authentication algorithms determine the security mechanism. It is thus necessary to consider identity management and access control as a whole for the purpose of high security and low cost.

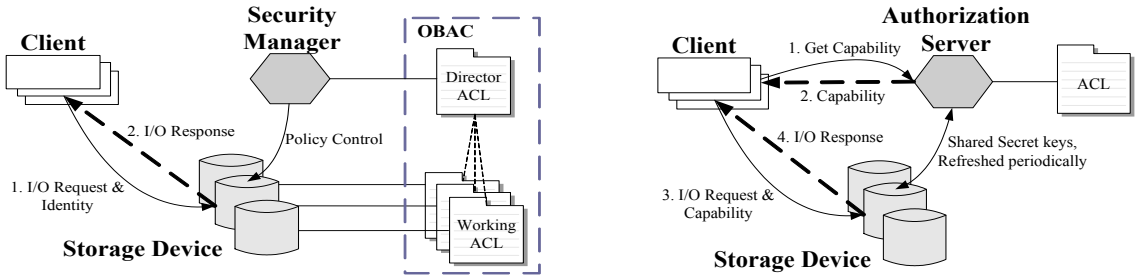
Unfortunately, existing security schemes [9]–[20] for large-scale storage systems are ill-prepared for addressing the above challenges. A key reason lies in the confusion between the security-specific metadata and the common file-system metadata. The former specifies authorization information, encryption keys, data access logging, etc, while the latter refers to location and length information of data, system configuration, etc. The two obvious differences between the above two kinds of metadata are their respective users and lifetimes. A file server or administrator authorizes application clients according to the security-specific metadata, such as authorization information, while users locate and access data by the common file-system metadata, such as location and length of the data. For security purposes, during the whole lifetime of the data, the security-specific metadata is updated frequently, for example, revoking authorization and refreshing keys regularly, while the common file-system metadata, such as location of the data, usually does not change. However, these key differences have been largely ignored in the design of the current security schemes, which results in the following inherent limitations.

Current large-scale storage systems have largely ignored security. The decoupled design of large-scale storage systems that separates metadata path from data path to enable direct interaction between clients and devices has not differentiated the security-specific metadata from the common file-system metadata. Consequently, storage devices, i.e., the users of the security-specific metadata, do not possess any explicit knowledge of access privileges and authorizations, meaning that, in order to authorize a client, the storage devices have to acquire authorization information from MDS or the client who has acquired a capability from MDS. Given the sheer scale of the systems under our study, this imposes an unacceptable overhead on MDS. To make things worse, the security-specific metadata is updated frequently, making it impractical for servers to generate and return that many capabilities in a timely manner, especially in HPC storage systems.

There exist redundancies and loopholes in current security mechanisms for large-scale storage systems. Existing large-scale storage systems authenticate clients at a centralized authorization server by utilizing an existing security infrastructure, such as Kerberos [21]. The authorization server grants the client access to the devices and then the devices enforce decentralized access controls, thus separating identity management from access control. This separation makes the system vulnerable to security attacks and incurs additional cost of access control.

Most of the current security schemes have ignored the complexity and scalability issue of security administration. Capability-based security (CapSec) schemes [9]–[19] widely used in most of the current security solutions maintain an access control list (ACL) at a centralized authorization server. Given the sheer scale of the systems under our study, this ACL can become too large and complex to maintain easily and efficiently. Identity key schemes [19], [20], which store the role-based access control list along with each object on the devices, reduce the complexity of security administration to a certain extent in an environment with a large number of clients. Nevertheless, as the number of and amount of data on the devices further increase as is the technological trend, data update (e. g., write operations) will still result in an enormous number of permission operations.

The traditional access control provided by ad-hoc, single-purpose systems has become outdated and is being replaced by the identity-based access control, as the world



(a) IDEAS (b) CapSec

Figure 1. IDEAS architecture VS. CapSec architecture.

is gradually becoming identity based (Phil Libin[1], 2006). Identity determines what you are and what you can do. An identity-based access control system would not only eliminate a number of passwords and user accounts, but also achieve a centralized management of network security. As a result, the U.S. government issued Homeland Security Presidential Directive number 12 (HSPD-12) in 2004, which mandated that all federal employees and contractors use a single, secure credential for access to sensitive physical and IT resources. Soon thereafter, the U.S. National Institute for Standards and Technology (NIST) published Federal Information Processing Standard 201 (FIPS 201) to address issues related to secure processes and attributes for the issuance, use, and interoperability of the identification credentials. It was proposed that all U.S. federal agencies start issuing interoperable identity cards to all employees by Oct. 27, 2007. In this paper, we attempt to merge identity management with access control to improve security, convenience and total cost of access control by eliminating the aforementioned redundancies and loopholes in the decoupled designs of parallel file systems and large-scale storage systems. The main contributions of this paper include:

- It develops IDEAS, an *ID*-based s*E*curity *A*rchitecture for large-scale and high-performance *S*torage systems, designed to improve security, convenience and total cost of access control (Section 2). IDEAS authenticates users at each I/O node by using a single-identity certificate without the service of a centralized security server and enforces access control by using an object-based access control (OBAC) model.
- It proposes an object-based access control (OBAC) model (Section 3), which is designed to address the complexity and scalability issue of security administration in large-scale and high-performance storage systems. A generic definition and a formal description of OBAC are presented. The access control rules for OBAC, namely, the PIPS (Proximity, Inheritance, Priority, Sharing) rules, proposed in this paper can be used as the basis for establishing a testing and evaluation criteria for securing general large-scale storage systems.
- It discusses the issue of how to identify and authenticate a large number of users with the state-of-the-art cryptographic solutions. The looming crisis of PKI, a widely used technology for authentication in today's information security area is discussed and potential alternative technologies to PKI are suggested (Section 4).
- It extends the T10 OSD (object-based storage device) standard [22] to support IDEAS and implement an IDEAS prototype in the HUST OSD project [12], a prototype implementation of the T10 standard in HUST (Huazhong University of Science and Technology). Experiments with both CapSec and IDEAS show that IDEAS significantly outperforms CapSec in terms of latency for key security-related operations, by a speedup factor of 1.81 and 2.22 for the frequent read and write operations respectively and by a factor of 1.65, 1.22, and 0.52 for the infrequent create, delete and chmod operations respectively. IDEAS completely removes the performance bottleneck caused by security overhead and drastically improves the scalability on object-based storage systems (OBS) while achieving higher security than the conventional CapSec solution (Section 5).

2. IDEAS Architecture

In an IDEAS storage system, shown in Figure 1(a), each user and component (e.g., storage device and security manager) has a universal identifier ID and can play a specific role during a given period of time. Each identifier includes a designation of the entity as a user or storage device; e.g., $ID_u = (user \parallel identifier)$. Access permissions are assigned to roles and any user that assumes a particular role is permitted to perform operations assigned to that role. User identifiers and the user-to-role association are assumed to be certified at regular intervals (say, monthly) by a TA (Trusted Authority) in our schemes. A certificate consists of the necessary certificate discriminator ID_{cert}, user identifier ID_u, role identifier ID_r, user public key generated using one of

1. The Founder and President of CoreStreet, one of the top companies providing smart credential and identity management technologies to governments and large corporations throughout the world.

the classic algorithms like RSA, expiration time and other optional items. Thus the certificate can be implemented using identity (ID) cards and are compliant with existing ID card systems.

IDEAS exploits the OBAC model to address the complexity and scalability issue of security administration (see Section 3) that has been ignored by most of the current security solutions for large-scale storage systems. The main idea behind the OBAC model is to associate ACLs (Access Control Lists) with objects and allow ACL inheritance. IDEAS stores *working ACLs* at storage devices and *director ACLs* at the security manager. The storage devices perform access control based on users' identities and the corresponding working ACLs, along with each stored object. The director ACL keeps a replica of all working ACLs and ensures the consistency of the ACLs distributed among multiple devices. By maintaining a director-ACL database at a centralized security manager, frequent changes to ACLs can be easily handled. All ACL changes will first go to the security manager and the security manager will propagate it to other storage servers.

Figure 2 describes the operations involved in creating and deleting (Line 1), reading and writing (Line 2) as well as modifying privileges (Line 3) in an IDEAS storage system.

1) **Create and Delete.** Creating or deleting an object results in the creation or deletion of the object itself, and the director and working ACLs for that object. Creating or deleting a single director ACL entails creating or deleting only one database record. When the director ACLs are stored as metadata, the cost of creating or deleting the director ACLs can be ignored at least for security purposes because creating or deleting database records for the director ACLs as a concomitant can be performed along with metadata operations.

2) **Read and Write.** For common read and write operations, users can directly interact with any storage servers by using a single certificate and the security manager can be completely offline, which minimizes the load on the security manager and reduces the impact of possible failure of the security manager on the system.

3) **Modify Privileges.** For the purpose of keeping ACLs consistent for all striped objects that belong to a single larger object, the changes to the ACLs of the latter object will first go to the security manager that will then propagate it to other storage servers. It should be noted that modifying privileges is a relatively infrequent event because changes to role permissions (i.e., to the role-based access control list) are infrequent relative to changes to role memberships. Such a scheme also reduces the workload of the administrator because, for each object, an administrator need only send a privilege command to the security manager but need not concern whether that object is striped onto multiple storage servers.

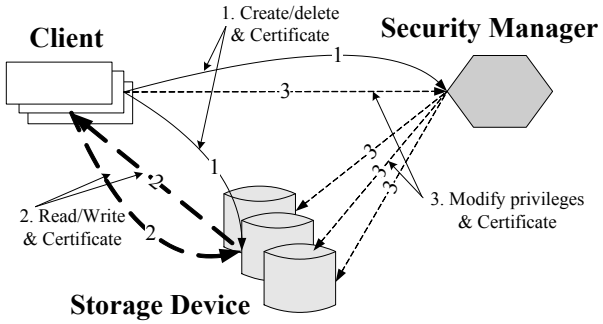


Figure 2. Operations for creating and deleting, reading and writing, as well as modifying privileges.

In contrast to the CapSec architecture, shown in Figure 1(b), that has been widely used in current security schemes, IDEAS not only improves the security, performance and scalability but also reduces the complexity of security administration for large-scale and high-performance storage systems, as explained next.

The CapSec scheme maintains ACLs at a centralized "authorization server". In order to access an object, a user must acquire a capability that grants permissions to the user from the server for each object that she wants to access. Thus, the centralized "authorization server" authenticates the user and authorizes the request while the storage devices validate the capabilities and perform requests permitted by the capabilities. As a result, identity management is separated from access control, which results in unavoidable security risks and additional cost of access control. For example, the capabilities and the key hierarchy that protects the capabilities increase the number of attack points because the shared secret keys in the key hierarchy can be revealed and malicious users can tamper with or forge the capabilities cached at user hosts or transmitted over insecure networks. Also, as applications scale to use tens or hundreds of thousands of nodes, the authorization server becomes a severe bottleneck for data access.

In IDEAS storage systems, on the other hand, authentication and authorization are performed at the storage devices. A user can directly interact with any device attached to the client-network by using a single-identity certificate without the service of a centralized security manager, which shortens the data access path and reduces the load on the security manager. Since the user has to prove her identity by a certificate (such as a smart card or passport), which is secured against physical or electronic forgery attempts and the storage devices, that is, data providers themselves determine the user's identity and specify what the user is allowed to do by the locally stored privilege information, the security of IDEAS storage systems has been significantly improved over that of CapSec storage systems that use capabilities to deliver privilege

information.

3. Object-based Access Control

Object-based access control (OBAC) is designed to address the complexity and scalability issue of security administration in large-scale and high-performance storage systems. The main idea behind OBAC, associating ACLs with objects and allowing ACLs inheritance, has been widely used in many ACL mechanisms, e.g., POSIX [23] and Windows NT [24] . In order to meet the security need of massive and frequently changing data in the information systems, Yang, et al. [25] have applied OBAC to electrical power systems, and Li, et al. [26] have applied OBAC to complex process information management. But these OBAC systems have been developed by a variety of organizations, with no commonly agreed upon definition or recognition in formal standards. As a core component of IDEAS, it is important for OBAC to be clearly defined in a common and formal way in this paper. In what follows we first discuss the shortcomings of the traditional access control models in the design for securing a large-scale and high-performance storage system, and then describe the fundamental features and access control rules of OBAC. A formal description of OBAC is given in Appendix A.

3.1. Traditional Access Control Models

The Trusted Computer System Evaluation Criteria (TCSEC [27]) specifies two types of access control: Discretionary Access Controls (DAC) and Mandatory Access Controls (MAC). However, both DAC and MAC are inadequate for large commercial applications [28]. Role based access control (RBAC) is an alternative to traditional DAC and MAC policies that is attracting increasing attention particularly for commercial applications. By mapping an organization's structure to a set of roles and operations that may be performed by these roles, RBAC can reduce the complexity and cost of security administration in large commercial applications.

Although RBAC is powerful in its ability to express the complex relationships between individuals and their access rights and address the complexity issue of security administration in large commercial applications, it is insufficient for implementing permission scenarios that usually occur on a large-scale and high-performance storage system. This is because, besides maintaining the relationships between roles and users, the security administrator must grant an enormous number of privileges on the large data sets to a limited number of roles. In addition, RBAC has not defined the forms of access privilege organization. We consider two common forms of access privilege organization: access control matrix and access control list (ACL). The former for large-scale storage systems can be too large and complex to maintain and operate efficiently and economically. With the latter, granting accesses to objects can be performed by assigning ACLs to these objects and can overload the security administrator when there are a large number of protected objects. In an HPC storage system that can create a large number of sensitive objects instantaneously, it is also impractical to create and assign a large number of ACLs in a timely manner.

3.2. Fundamental OBAC Features

We follow the basic idea of the access control list mechanism and propose an OBAC model to address the aforementioned shortcomings of traditional access control models. It should be noted that OBAC can be viewed as an independent component of access control, coexisting with DAC, MAC and RBAC when appropriate. Three fundamental OBAC features are summarized below.

1) **ACL's association with an object.** With OBAC, each object is associated with an access control list, which is made up of a list of access control entries (ACEs) that specify who is allowed what access to that object. In large-scale and high-performance storage systems, the number of objects far exceeds that of users and roles combined and changes to a typical object are more frequent than those to either a user or a role, making permission operations to objects the overwhelmingly dominant operations for security administration. It is significant for large-scale and high-performance storage systems to associate ACLs with objects because performance is becoming more important than ever. Once an object is located the corresponding ACLs will be located, thus access control and permission management will become more direct and efficient.

2) **ACL inheritance.** ACLs can be set to inherit from their parents, that is, the ACL of the parent will be applied to the child. The inheritance of ACLs has been addressed in many settings ranging from object-oriented databases to distributed systems. The main motivation for the ACL inheritance is to simplify management of ACLs in large hierarchical systems. There can be tens of millions of protected objects in large-scale storage systems, and every single object needs to be assigned an ACL. ACL inheritance allows an administrator to set permissions for a parent that can be applied to a set of child objects (possibly recursively), instead of manually setting permissions one object at a time. There are two types of ACL inheritance [24]: static inheritance and dynamic inheritance. Static inheritance occurs when an object is created or a new ACL is written to an object due to an ACL change. The inherited access control entries are copied from a parent into the ACL of a child object. With static inheritance, only the ACL on the object itself must be evaluated for most access checks. Dynamic inheritance occurs during access checks. The inherited access control

entries on a parent are automatically inherited to the child. With dynamic inheritance, multiple ACLs must be evaluated in access checks.

As a result, static inheritance incurs lower runtime costs than dynamic inheritance. However, compared to the latter, static inheritance results in more ACLs being modified when access control changes, and generates larger ACLs. So static inheritance is preferred when the performance of propagating ACL changes is less critical than the speed of an access check. Dynamic inheritance presents a simple and intuitive access control model that offers good manageability and low storage costs, at the cost of checking access on multiple parent objects.

3) **ACE priority.** In OBAC, privileges may be granted to either users or groups, that is, an ACL may contain an ACL entry for a user or a group. A user can be a member in more than one group, implying that one user entry and more than one group entry can match in access checks. Further, *positive* ACEs and *negative* ACEs may be allowed in a system, where the former grants access and the latter denies access. Finally, with dynamic ACL inheritance, an ACE matching the same user or group can appear in both the inheriting ACL and the inherited ACL. However, only a single entry can determine access. To avoid the potential inconsistencies in multiple matching ACEs, we must establish an order of priority for ACEs. For example, we can place negative ACEs before positive ACEs, so as to make deny entries take precedence over grant entries.

3.3. The PIPS Rules for OBAC

We define four basic access control rules for OBAC, namely, Proximity, Inheritance, Priority and Sharing, or the PIPS rules. These rules can be applied to not only the IDEAS design but also any other large-scale and high-performance security systems that employ OBAC.

Proximity - Proximity means placing access privilege information as close to protected objects as possible. This rule ensures that access control decisions can be made as soon as possible without the additional costs of locating and fetching the required access privilege information.

Inheritance - Inheritance means that an object can automatically obtain the corresponding permissions granted to its parent without having to manually assign these permissions to it. This rule ensures that operations of object creation can be performed quickly in a high-performance system, which can generate a large number of objects in a timely manner and each object must be assigned the corresponding permissions.

Priority - Priority refers to subject and permission priority. This rule ensures that an arbitration decision can be made when an inconsistency occurs as a result of privilege inheritance and sharing.

Sharing - Sharing makes it possible to reduce the complexity of security administration in large-scale systems with an enormous amount of users and other protected resources. For example, users associated with a particular role can share the same rights and responsibilities, while a child object can share her parent's rights. Thus the security administrator does not need to assign, change, and revoke privileges for each user or child object.

4. IDEAS Design

IDEAS provides strong, manageable and low-cost security for large-scale and high-performance storage systems by merging identity management with access control. In this section, we discuss the issue of how to identify and authenticate a large number of users with the state-of-the-art cryptographic solutions, and then detail the design of access control lists and the revocation mechanisms in IDEAS.

4.1. Identifying and Authenticating an Enormous Number of Clients

Large-scale storage systems may service millions of clients. The authentication technologies for such systems must be scalable and provide a simple authentication process. Authentication systems are an essential component in cyber security in general. On the other hand, the key of authentication systems is to establish a rational digital signature, which in turn relies on an effective key management. Thus cyber security must solve the issue of large-scale key management because there are an enormous number of clients to identify and authenticate in cyberspace.

There are three potential solutions: public key infrastructure (PKI), identity-based encryption (IBE) and combined public key (CPK) [29]. The PKI technology first presents the notion of a third-party authentication and meets the requirement for user identification, authentication, and non-repudiation in cyber security. Though PKI can be used as a solution to many security problems, such as secure e-mail, e-business and e-banking, it is not an ideal authentication technology. In PKI systems, a public key is bound to the key-holder's name by a certificate authority (CA). But the trustworthiness of CA has long been considered illogical. Many current PKI implementations employ a hierarchical CA model where, since a single CA can service only a limited number of users, PKI increases the scale of key management by adding CAs, which in turn gives rise to the issue of agency expansion and increasing network traffic.

The IBE scheme employs the identification of a user's identity as a public key or derives the public key from the user's identity. As a result, IBE can work without

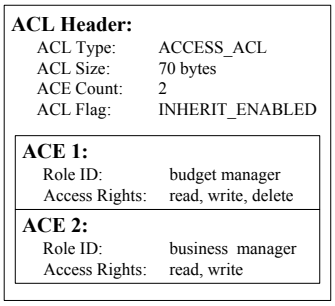


Figure 3. An sample ACL on an object with two entries (ACEs).

the support of a CA hierarchy, thereby eliminating the increasing demands on bandwidth incurred by constantly expanded CA facilities. However, IBE systems can not work without an on-line public parameter server (PPS), which provides IBE public parameters and policy information for an IBE private key generator (PKG). As a result, IBE is not a true two-party authentication process (i.e., devoid of a third party such as CA in PKI). CPK, one of the IBE variants, uses a small amount of seeds to produce an almost limitless amount of keys in order to meet the almost limitless demand for keys. CPK holds only a small number of public parameters, which can be stored in a tiny chip. Thus CPK will be able to obtain the necessary public parameters for a public key from the chip without the on-line PPS support. As a result, CPK fully implements peer-to-peer authentication.

In sum, IBE addresses the crisis of trust that confounds PKI and is considered a potential alternative technology to PKI, whereas CPK as one of the IBE variants can further solve the issue of lager-scale key management. A detailed description and comparison of the PKI, IBE and CPK systems is given in Appendix B, which will help provide insight into the design decision on the use of an appropriate key management scheme.

4.2. Access Control List

The design of ACLs based on OBAC for IDEAS systems is described blow.

4.2.1. ACL Structure. Two types of ACLs are proposed for two different purposes in our scheme, namely, *access ACL* that defines the usual access permissions of protected objects and *super ACL* that defines the permissions capable of inheritance by a child object from its parent object. Access ACLs can be associated with both objects and container objects that contain child objects, such as directories. Super ACLs can be associated with only container objects. Super ACLs for non-containers would be of no use, because no other system objects can be created inside non-containers.

A sample ACL with two access control entries (ACEs) is shown in Figure 3. An ACL is made up of a header and an arbitrary number of ACEs for different roles. The ACL type field specifies the type of this ACL and the ACL flag controls whether a child object dynamically inherits its parent object's super ACL in access checks. Although the OBAC model supports an ACL containing ACEs for users and groups (members in a group can also be grouped into a role), we only define the ACEs that grant access rights to specific roles for the purpose of reducing the cost of security administration incurred by the frequent changes for user access rights. ACE contains the identifier of the role it grants access to and an access mask that is a bit-field specifying the access rights.

4.2.2. ACL Inheritance. Large-scale storage systems may store tens of millions of files or objects, with each needing to be assigned an ACL. Storing and managing so many ACLs can cost considerable storage space and impose significant, if not prohibitive, workload on the security administrator. In an HPC storage system, it is also impractical to manually create an ACL for each of its large number of objects in a timely manner. ACL inheritance allows a child object to utilize its parent's ACL, instead of storing its own ACL and thus allows an administrator to focus on setting permissions for entire branches of the tree in large hierarchical systems, instead of manually setting permissions one object at a time.

In our approach, ACLs can be set to inherit from their parents. What this means is that the ACL of the child will utilize the parent's super ACL. For example, the permissions for "/jim/foo" can utilize the super ACL associated with "/jim". The ACL flags, INHERITANCE_ENABLED and INHERITANCE_DISABLED, indicates whether it inherits its parent's super ACL or not. In the absence of ACLs associated with an object, the object will inherit its parent's super ACL by default, and in this case, if no super ACLs are associated with the parent, the object is inaccessible.

ACL inheritance occurs only when the ACL is set to inherit from the parent object and no ACE is hit in the current ACL. Consider, for example, the inheritance that occurs in a hierarchical file system. If inheritance is enabled, when the access check process reaches the end of the current list, it will start traversing the super ACL of the parent folder to find a matching ACE. Therefore, the object will have the parent's super ACL applied to itself, automatically. If the folder also has inheritance, the access check process will traverse that folder too. This can continue all the way until the access check process hits a folder that has inheritance disabled, or it hits the root of the device.

In additional, only super ACLs are inherited in access checks, so changes to an access ACL of the parent apply only to the parent itself but not to the children. Since the

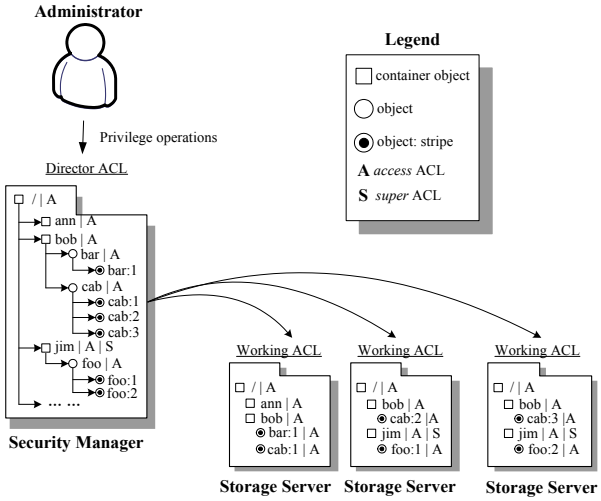


Figure 4. The ACL storage sketch map.

ACL of the child has precedence over that of the parent, only the ACEs that appear in the parent's super ACL but not in the child's access ACL can be applied to the child.

4.2.3. ACL Storage. Figure 4 shows an ACL storage sketch map in our system. The security manager stores director ACLs, which contain access ACLs and super ACLs along with objects and container objects in the form of database records, and therefore all changes to the director ACLs will be mapped to the database records. Changes to a single object's permissions affect only one record. Although a change to the specific role's permissions results in simultaneously updating multiple database records, it is infrequent because an ACL only stores the role-based ACEs and changes to role permissions are infrequent as compared to changes to role memberships. When the director ACLs are stored as a kind of metadata along with other metadata such as the mapping of files to sets of objects, file system namespace information, and configuration information about the storage system itself, the existing policies applied to metadata can apply to the director ACLs.

The working ACLs, which contain access ACLs and super ACLs, can be stored as object attributes. Once an object is located, so is the object's working ACL. Since a container object's permissions may affect the child objects belonging to it, such as read directory and write directory, and the super ACL of the container object may apply to the child objects, the storage devices that store the child objects also store the access and super ACLs of the container object. In this case, the container object can be mapped to a directory or a partition in the storage devices.

As aforementioned, traversing the ACLs of all the objects located in the devices due to changes to the specific role's permissions is infrequent as compared to simultaneously updating a single object's attributes due to changes to the object's permissions.

4.3. Revocation

Revocation can be achieved in the following ways:

1) **Certificate expiry.** Each certificate has a lifetime defined by the expiry time. When the certificate held by a user expires, the user has to acquire a new certificate to declare its own identity.

2) **Identity revocation list.** Unlike the traditional certificate based systems,such as PKI, that revoke certificates by issuing certificate revocation lists (CRLs), IDEAS explicitly revokes identities by issuing identity revocation lists (IRLs). It is particularly useful in identity-based systems, where revocation can be integrated into authentication systems and thus eliminate the need to ransack all over the Internet to confirm the validity of a certificate. For example, revoking accounts is an essential operation of a banking system. The identity revocation lists can maintain information regarding revoked user identifiers and a particular role from a specific organization, and can be periodically issued by the security manager.

3) **ACL.** We can change an object's access privileges by changing the ACLs associated with that object. Role privileges can be changed by changing the ACLs that contain ACEs for the specific role. Although it costs more, a change to the role privileges is relatively infrequent.

4.4. Discussion

Since identification technologies and authentication algorithms determine the security mechanism, IDEAS merges identity management with access control to improve security, convenience and total cost of access control by eliminating those redundancies and loopholes introduced by the decoupled design of data and metadata in parallel file systems and large-scale storage systems. IDEAS can work with not only the traditional PKI systems but also the state-of-the-art authentication solutions, such as the IBE and CPK systems. The design of ACLs in IDEAS is based on the OBAC model that associates ACLs with objects, allows ACLs to be inherited, and addresses the complexity and scalability issue of security administration in large-scale storage systems. The identity-based architecture of IDEAS allows identity-based revocation. Besides the traditional revocation mechanism such as expiry and ACL, a novel revocation mechanism, identity revocation is suggested for identity-based systems to improve the efficiency of revocation. The design and study of these solutions will provide valuable insight and guidance for practitioners/designers to design efficient access control mechanisms for large-scale and high-performance systems.

5. Implementation and Evaluation

We propose several extensions to the current T10 OSD standard to support IDEAS and have implemented an IDEAS prototype in the HUST OSD. Our new OSD implementation supports not only IDEAS but also CapSec that has been defined in the T10 OSD standard. In this section, we first present a detailed IDEAS implementation. And then we describe the system assumptions of the IDEAS implementation and discuss the security of the IDEAS system. Finally, we present experimental results obtained from the IDEAS prototype to evaluate the overhead of security, as well as system scalability of IDEAS.

5.1. Implementation Details

The current IDEAS implementation consists of three components: MDS, OSD, and clients. Besides metadata management, MDS also implements the functionalities of security manager, in which director ACLs are stored as LDAP (Lightweight Directory Access Protocol) database records along with other metadata. Creating or deleting the director ACLs can be performed along with metadata operations. All operations modifying privilege will first go to MDS where they will be propagated to relevant OSDs. Working ACLs are stored as object attributes along with objects. As aforementioned, in order to reduce the cost of security administration incurred by the frequent changes to user access rights, each director or working ACL only stores the ACEs for specific roles.

Authentication in the current IDEAS implementation is based on IBE, which requires each user and component to obtain an identity certificate from a TA that certifies the user identifier and user-to-role association. The IBE private key is obtained after authenticating to the TA that calculates private keys for users. Unlike PKI systems, in which keys are generated randomly, IBE systems allow keys to be calculated from an identity. If a recipient obtains the IBE public parameters, she can calculate the sender's public key from its identity. Before establishing access to system resources, an application client authenticates herself to MDS and any OSD that she wishes to access. The two parties in a conversation verify each other's identity by sharing a secret session key, which has been created in the logging process using an ID-based authenticated key agreement protocol [30]. As a rule, public key cryptography is orders of magnitude slower than shared key cryptography, but the ID-based authentication usually occurs only in the logging process. The raw speed and effect to the system of the cryptographic algorithms used by our system are evaluated and details of this evaluation are presented in Appendix C. The evaluation indicates that the RSA and IBE signatures take the most amount of running time, roughly $5.8ms$ and $16.4ms$ respectively, while with an input of 512 bits, roughly the length of a command, the AES and HMAC-SHA1 algorithms, which provide the confidentiality and integrity protection to commands in the IDEAS implementation, incur latencies of only $17\mu s$ and $14\mu s$ respectively. Such overheads are so sufficiently insignificant that they should not constitute the primary security cost.

Now, the client creates a shared key K_{CM} with MDS and a shared key K_{CD} with each OSD that she wishes to access. K_{CM} is used to cryptographically harden the messages between MDS and the client in the subsequent conversations, while K_{CD} is used to protect the OSD commands from various network attacks, similar to the use of the capability key for securing an OSD command with the CMDRSP[2] security method in the T10 OSD standard. In the current IDEAS implementation, the integrity of the CDB (command descriptor block), returned status and exceptional condition for each command is validated by the shared key K_{CD}.

We extend the T10 OSD command set to support IDEAS. The authenticated key agreement protocol is accomplished via the SET SESSION KEY command, which negotiates a secret session key between OSDs and clients. The SET ACCESS ENTRIES command retrieves and sets the ACEs of protected objects.

5.2. System Assumptions

The security of the IDEAS implementation is built on the following system assumptions.

1) **MDS.** We assume that MDS is a trusted component. It can properly serve metadata to clients, provide integrity for the stored metadata, safely store access control lists and not be controlled by an adversary.

2) **OSD.** We also assume that OSD is a trusted component. It can properly serve data to clients, provide integrity for the stored data, safely store access control lists and not be controlled by an adversary.

3) **Clients.** Each client acts on behalf of a number of users. We assume that the users trust their own operating system to protect them from malicious users on the same machine who may access data in local memory. However, clients or end users are not assumed to be trusted because they can perform all kinds of active and passive attacks. For example, a legitimate client or user can attempt to impersonate other legitimate clients, and an adversary can break into other legitimate clients to perform illegitimate access.

4) **Communication Links.** The communication links are assumed to be completely insecure. An adversary can perform active and passive attacks such as eavesdropping, masquerading, replaying, and modifying data.

2. One of the three security methods defined in the T10 OSD standard.

5) **Cryptographic Primitives and Security Protocols.** Our implementation utilizes two cryptographic primitives, the AES and HMAC-SHA1 algorithms, and one key agreement protocol, the Yuan and Li's ID-based authenticated key agreement protocol [30]. For that, we assume that these algorithms and protocol are sufficiently hard to break.

These assumptions do not impose more stringent requirements than those of the CapSec scheme. And this is also the case in many other parallel file systems. However, in contrast to CapSec, which utilizes a triple security protocol to achieve access control and a complex key hierarchy to protect the capabilities, IDEAS exposes much fewer attack points. In IDEAS, once a user's identity is validated, storage devices can directly determine access controls by the locally stored privilege information, thus an IDEAS storage system offers much better security than a CapSec storage system.

5.3. Experimental Setup

IDEAS is designed to address the complexity and scalability issue of security administration for large-scale storage systems or next-generation storage systems, which may hold hundreds of metadata servers and hundreds of thousands of storage devices, and service millions of clients. It is thus impractical for us to experiment on such a storage system in its full scale. However, since IDEAS' design goals are to decentralize the security administration and remove performance bottlenecks of the conventional schemes, it will be sufficient to assess IDEAS's scalability and efficiency by evaluating the following key and meaningful metrics on our small IDEAS prototype: (1) various overheads associated with IDEAS systems; (2) MDS's idle CPU time under a high-bandwidth workload; and 3) system scalability under a metadata-intensive workload. We compare the security overhead of IDEAS to that of CapSec to assess the performance benefits of IDEAS systems, and compare MDS's idle CPU time of the IDEAS implementation with that of the non-secure and CapSec implementations to evaluate how much security overhead is incurred when the IDEAS and CapSec schemes are added to an object-based storage system respectively. The analysis of system scalability will provide insight into the system's performance bottlenecks and how IDEAS removes such bottlenecks.

Our experiments were conducted on 3 to 17 hosts, each with one Intel Xeon 3.0 GHz processor and a total of 512 MB DDR-SDRAM physical memory. In addition, each node is connected to a Highpoint Rocket 2240 Raid controller attached to 15 SATA disks (7200RPM, 300GB each). All machines run Fedora Core 4 (kernel version 2.6.12) and are connected by a 1-Gbits Ethernet. In each experiment, one machine acts as MDS, while others act as OSDs or clients. For CapSec, the CMDRSP security method was used in our experiments because this method provides the same security as IDEAS does.

5.4. Latency Breakdown

We ran a set of micro benchmarks on the IDEAS and CapSec implementations and measured the latency of each operation in order to evaluate the various overheads associated with our OSD implementation and the performance benefits of IDEAS when compared to CapSec. The experiment was performed on a one-client to one-OSD system. We assumed that the client had authenticated herself to the MDS and OSD, and negotiated a shared key with the latter two respectively. All the latency benchmarks were run on a collection of 512 files, each of size 4 KB. In each benchmark, a fixed filesystem operation (e.g., read, write, or chmod) was performed on each of the files in a randomized order. For each operation, the MDS, OSD and client drivers were instrumented to report the time spent in fine-grained sub-operations shown in Table 1.

Table 2 summarizes the latency breakdown of the various operations for the IDEAS and CapSec systems and the percentage of the latencies for the sub-operations in relation to that for the MDS or OSD commands. An empty cell in the table denotes a sub-operation that is not performed. As observed in the table, the security overhead of IDEAS, including MDS and OSD command building and validation, is comparable with that of CapSec. However, the performance benefits of IDEAS come from the significantly reduced round trips from the client to the MDS. Since a client wishing to access OSDs has to acquire a capability from the MDS in the CapSec system, all the common operations, such as create, delete, read and write, each involve a round trip from the client to the MDS as well as a round trip from the client to the OSD. In contrast, the two IDEAS-induced round trips from the client to the MDS and OSD can be performed simultaneously for the create and delete operations, which will be completed after the longer of the two round trips finishes. As a result, IDEAS speedups CapSec by a factor of 1.65 and 1.22 for the create and delete operations respectively. More importantly, for the much more frequent read and write operations, IDEAS requires only one round trip from the client to the OSD, resulting in a speedup of 1.81 and 2.22 respectively over CapSec. For the chmod operation, IDEAS requires two round trips from the client to the MDS and OSD respectively, while CapSec requires only one round trip from the client to the MDS, because in the CapSec system the chmod operation only needs to modify the privilege information in the MDS. But it should be noted that the chmod operation is much less frequent than the common operations, such as read and write.

If the client cached the unexpired capabilities that can be used in the subsequent accesses to the OSD, the CapSec

Table 1. Operation breakdown

Command	*Sub-operation*	*Overhead*
MDS	MDS command building (MdsCmdBuild)	MDS command encapsulation and encryption
	Communication to MDS (CommToMds)	Latency that an MDS command is transmitted from clients to MDS; and Latency that metadata is returned from MDS to clients
	MDS command validation (MdsCmdValid)	MDS command decryption and privilege verification
	Capability generation (CapGen)	
	Database access	
OSD	OSD command building (OsdCmdBuild)	OSD command encapsulation and MAC computation (to compute the integrity check value of the command)
	Communication to OSD (CommToOsd)	Latency that an OSD command is transmitted from clients to OSD; and Latency that results are returned from OSD to clients; and Latency that data is transmitted
	OSD command validation (OsdCmdValid)	MAC computation and privilege verification for IDEAS; or MAC computation and capability reconstruction for CapSec
	Disk access	

Table 2. Latency breakdown of various operations for the IDEAS and CapSec systems.

Latency (μs)	*Create*		*Delete*		*Read*		*Write*		*Chmod*	
	IDEAS	*CapSec*	*IDEAS*	*CapSec*	*IDEAS*	*CapSec*	*IDEAS*	*CapSec*	*IDEAS*	*CapSec*
MdsCmdBuild	78(7%)	78(7%)	78(3%)	78(2%)		78(10%)		78(7%)	78(10%)	78(10%)
CommToMds	227(21%)	227(21%)	227(7%)	227(7%)		227(29%)		227(21%)	227(30%)	227(30%)
MdsCmdValid	46(4%)	46(4%)	46(1%)	46(1%)		46(6%)		46(4%)	46(6%)	46(6%)
CapGen		45(4%)		45(1%)		45(6%)		45(4%)		
DatabaseAccess	710(67%)	710(64%)	2750(89%)	2750(87%)		391(50%)		710(64%)	405(54%)	405(54%)
MDSCommand	1061	1106	3101	3146		787		1106	756	756
OsdCmdBuild	62(10%)	62(10%)	62(10%)	62(10%)	62(6%)	62(6%)	62(7%)	62(6%)	62(9%)	
CommToOsd	258(43%)	258(40%)	258(44%)	258(41%)	571(56%)	571(53%)	329(35%)	329(33%)	258(36%)	
OsdCmdValid	211(35%)	253(39%)	211(36%)	253(40%)	211(21%)	253(24%)	211(22%)	253(26%)	211(30%)	
DiskAccess	73(12%)	73(11%)	62(10%)	62(10%)	184(18%)	184(17%)	340(36%)	340(35%)	180(25%)	
OSDCommand	604	646	593	635	1028	1070	942	984	711	
OperationCost	1061	1752	3101	3781	1028	1857	942	2090	1467	756
Speedup	**1.65**		**1.22**		**1.81**		**2.22**		**0.52**	

system will require only one round trip from the client to the OSD, which costs $1070\mu s$ for the read operation and $984\mu s$ for the write operation. Such costs are comparable with the overhead of the read and write operations in IDEAS. However, once the capabilities expire, the client has to request new capabilities, which requires two additional round trips from the client to the MDS and OSD respectively. Although the administrator can extend the lifetime of the capabilities to improve the system performance, for example, increases the expiry time of the capabilities or delays updating the shared keys between the MDS and OSD, it renders the system more vulnerable to security attacks.

5.5. Scalability

We ran a benchmark to measure MDS's idle CPU time of the non-secure, CapSec and IDEAS implementations with various numbers of OSDs and clients. The purpose of this experiment is to evaluate how much security overhead is imposed on MDS under a high-bandwidth workload when the IDEAS and CapSec schemes are added to an object-based storage system respectively. We ran the benchmark with 1 through 8 OSDs and clients. Each client read and wrote files on an OSD and each OSD was accessed by exactly 1 client. In each run, each client created 512 files, each of size 256 KB, on an OSD and sequentially read and wrote these files in 64 KB chunks. In order to make our experiments more realistic, we setup a cache of 512 entries for file metadata and a cache of 32 entries for capabilities in each client; that is, the clients can hit most of file metadata in cache, but miss most requests for capabilities in cache, which can expire or become invalid due to key update in reality.

Figure 5 shows that the average percentage of idle CPU time on the MDS declines with the number of clients/OSDs. However, in contrast to the non-secure system, IDEAS does not impose any additional overhead on the MDS because no capabilities are required for the read and write operations in the IDEAS system. Thus the MDS's idle CPU time of the IDEAS system is comparable to that of the non-secure system. On the contrary, the MDS's idle CPU time with the CapSec system declines faster than that of the non-secure and IDEAS systems because the MDS must prepare a capability for each read or write request with CapSec. Note also that the MDS's idle CPU time for the write operation is lower than that for the read operation, because in our current system implementations the MDS must synchronously update the file length for each successful write command,

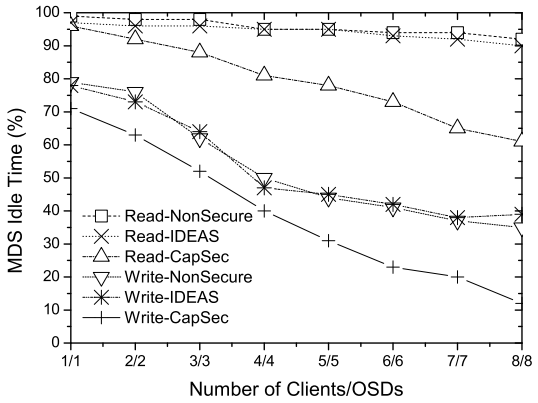


Figure 5. Average percentage of idle CPU time on the metadata server.

which not only increases the latency of a single write operation but also overloads the MDS when a larger number of write operations arise. This suggests that we can decentralize more metadata, such as file length and privilege information, from MDS to storage devices to improve the system scalability without impacting on the overall system management. Since the number of MDS requests for security purposes in the IDEAS system is significantly reduced, we expect IDEAS to be able to support a considerably larger number of clients than CapSec.

Given the fact that the scalability of object-based storage systems is largely influenced by the metadata server, we ran a metadata-intensive benchmark to measure the MDS latency in order to further study IDEAS's scalability. In this experiment, we ran multiple client processes on a total of 4 client machines. We varied the number of clients from 10 to 80, each issuing 100 create requests for different files, 100 read and 100 write requests for the same file at a time. To minimize the effect of different database deployments, our experimental setup allows the MDS to bypass more metadata requests to the database by caching frequently used metadata. Figure 6 charts the latency of MDS requests as a function of the number of requests, showing that both the average MDS read and write latencies are far lower than the average MDS create latency, because most of read and write requests can hit the same metadata since each request by a client is restricted to the same file.

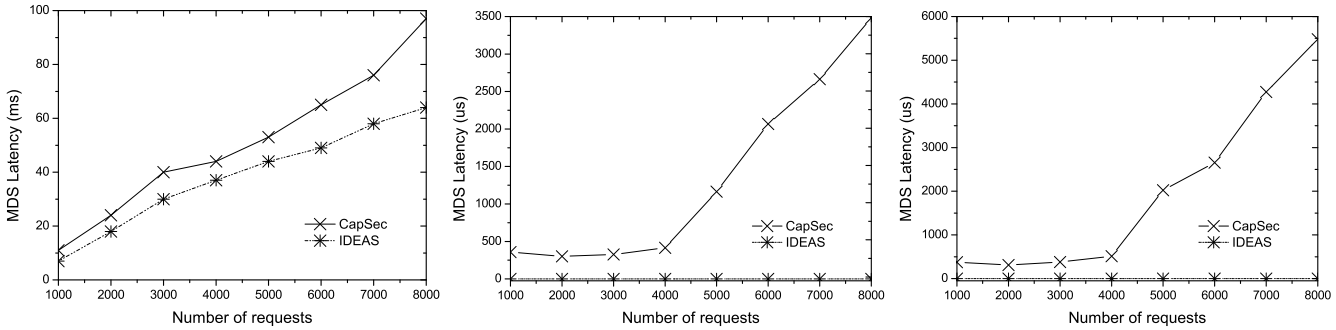
On the other hand, the average MDS create latency shown in Figure 6(a) for both IDEAS and CapSec ascends as the number of requests increases. However, with the same number of requests, the average MDS create latency for IDEAS is lower than that for CapSec because the MDS in the latter consumes a considerable amount of CPU cycles for the MAC computation to generate capabilities. It should also be noted that the MDS create requests are issued much less frequently than the read and write requests. For the frequent read and write requests shown in Figure 6(b) and 6(c), at a load of 4000 requests, the MDS with CapSec is saturated and becomes a bottleneck. With the number of requests larger than 4000, the MDS latencies soar quickly. On the contrary, IDEAS completely removes the bottleneck caused by security overhead, incurring a $zero$ latency, by avoiding capabilities for the read and write operations. Since the highly frequent security-specific requests are removed from MDS, the scalability of IDEAS has been drastically improved over that of CapSec.

6. Related Work

There have been numerous efforts to secure parallel and distributed storage systems, most of which are based on capabilities. CapSec storage systems maintain authorization information on a centralized authorization server, such as a security manager. The key differences between the security-specific and common file-system metadata, which IDEAS distinguishes and exploits for security and performance gains, have been largely ignored in these security schemes, which results in their inherent limitations in securing large-scale and high-performance storage systems.

The initial CapSec schemes are based on fine-grained capabilities, such as NASD [13], Azagury, et al. [10], Snapdragon [9], Snare [15], and the T10 OSD security protocol [22], which authorize access privileges at the granularity of a block or object, requiring the generation of tens of or even hundreds of millions of capabilities. This imposes a substantial overhead on the security manager, which has to be online and thus presents a central point of failure. If the security manager is down the entire system comes to a halt. LWFS [14] employs coarse-grained capabilities to grant access to a container of a group of objects, but constrains the granularity of access control. Moreover, the LWFS capabilities can only be verified by the authorization server that generated them. As a result, the additional overhead can quickly overload the authorization server.

Leung and Miller [18] propose an extended capability that authorizes I/O for any number of clients to any number of files that span any number of devices by using public-key cryptography. However, unlike IDEAS, which uses a role to denote a group of users having the same rights and responsibilities and stores role-based access control lists, extended capabilities identify a group of users having the same access permissions to a set of files and the files themselves via the root hash of a Merkle hash tree [31] constructed from the user IDs and file identifiers, and stores an access control matrix on MDS. However, the MDS's access control matrix with large-scale storage systems can become too large and complex to maintain and operate efficiently and economically, and



(a) Average MDS create latency. (b) Average MDS read latency. (c) Average MDS write latency.

Figure 6. An analysis of CapSec and IDEAS's scalability.

the heavy MAC computation and signature can quickly exhaust MDS.

SCARED [19] extends NASD to provide mutual authentication between clients and devices. It supports authentication based on capabilities (as in NASD) as well as identity keys. However, unlike IDEAS, which uses a identity certificate got from TA to certify a user's identity and role that the user can paly, the SCARED identity key is derived from a shared secret key between the administrator and the devices. In order to access multiple objects on multiple devices, a client has to acquire multiple identity keys; therefore, the total number of keys in the system does not decrease significantly, which is equal to the number of clients times the number of devices. Although the number of identity keys can be reduced by grouping the clients or devices, it is insecure in large-scale systems because all clients or devices within one group share one common secret that will be compromised as long as an attacker compromises a single device.

Kher and Kim [20] exploit RBAC in their system. In most of the cases, a client needs to acquire an identity key from the file manager, which can be used by the client to further derive role keys. Since the identity key is derived from the shared secret key between the file manager and the devices, the number of identity keys that clients have to contact the file manager to acquire is equal to that in the case of SCARED. By using the Diffie-Hellman Based Authentication (DHA) they reduce the number of keys to the number of clients plus the number of devices, which is less than the number of keys required by SCARED. Unlike SCARED, which stores identity-based access control lists along with each object, Kher and Kim store role-based access control lists with each object. It reduces the complexity of security administration in a large-scale system. However, like the aforementioned schemes, Kher and Kim do not address the design and implementation of role-based access control lists, a key to cope with the complexity and scalability issue of security administration in large-scale, demanding environment for which IDEAS was designed.

7. Conclusions

This paper describes IDEAS, an identity-based security architecture for large-scale and high-performance storage systems. By merging identity management with access control, IDEAS can improve security, convenience and total cost of access control for large-scale storage systems with millions of clients and hundreds of thousands of devices. IDEAS authenticates users at each I/O node by using a single-identity certificate without the service of a centralized security server and enforces access control mechanism by using an object-based access control (OBAC) model, which is designed to address the complexity and scalability issue of security administration in large-scale storage systems. The access control rules for OBAC, namely, the PIPS rules, proposed in this paper can be used as the basis for establishing a testing and evaluation criteria for securing general large-scale storage systems. We also discuss the issue of how to identify and authenticate a large number of users with the state-of-the-art cryptographic solutions and suggest the potential alternative technologies to the well-known PKI mechanism.

We extended the T10 OSD command set to support IDEAS and implemented an IDEAS prototype in the HUST OSD project. Experiments on the IDEAS prototype show that IDEAS is able to achieve high performance and scalability on object storage systems while achieving higher security than the conventional CapSec solution. In contrast to CapSec, IDEAS does not impose more stringent security requirements while exposing much fewer attack points. For the infrequent create, delete and chmod operations, IDEAS speedups CapSec by a factor of 1.65, 1.22 and 0.52 respectively, while for the frequent read and write operations, IDEAS achieves a speedup of 1.81 and 2.22 respectively. IDEAS completely removes the performance bottleneck of the system caused by security overhead, incurring a $zero$ latency, by avoiding capabilities for the read and write operations. As a result, the scalability of IDEAS has been drastically improved over

that of CapSec.

Appendix A.
Formal Description of OBAC

The main components of the OBAC model, i.e., element sets and relations, are defined in Figure 7. There are four distinctive sets of data elements called Objects (O), Subjects (S), Operations (OP) and Permissions (P). The OBAC model as a whole is fundamentally defined in terms of permissions being associated with objects. As such, once an object is located the corresponding permissions on that object will also be located. In addition, the diagram shows a set of Transactions (T), where each transaction is a mapping between an object and a matching subset of permissions that are assigned to the object.

Objects represent protected system resources, which is commonly referred to as protected objects within the computer system. For a system that implements OBAC, the objects can represent data objects, such as database tables and files, or resource objects, such as printers, CPU cycles, and disk space.

The concept of *subject* determines the basic policies and workload of security administration for the access control model. Since the ultimate purpose of any access control mechanism is to determine operations that a user can perform, the primary subject is a user. A user is defined as a human being. The concept of a *user* can be generalized to include machines, networks, or intelligent autonomous agents. For the purpose of reducing the workload of security administration, most access control models extend the concept of a user. Besides individual users, the set of subjects covered by OBAC also include a group, which is defined as a set of users that have the same responsibilities and authorities.

A *permission* is an approval or denial of a subject's operations. A positive permission grants the subject's accesses and a negative permission denies the subject's accesses. Although both positive and negative permissions can be deployed in an OBAC implementation, for simplicity reasons we only discuss a positive permission in this model. An *operation* is defined as executing one or more command functions for a subject. The types of operations greatly depend on the kind of system in which they will be implemented. For example, an operating system can perform read, write, delete, and execute operations; a relational database management system can perform SELECT, UPDATE, DELETE, and INSERT operations.

Figure 7 shows the *permission assignment (PA)* relation, a many-to-many relation. An object can have many permissions, and the same permission can be assigned to many objects. Each *transaction* is a mapping of one object to possibly many permissions, that is, during a transaction multiple permissions can match. However,

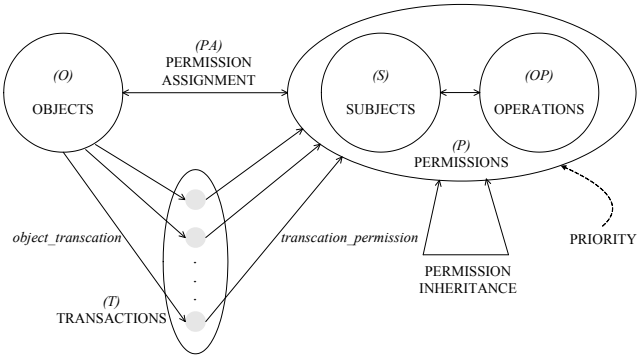


Figure 7. The OBAC model.

the priority of these permissions ensures that only one permission can determine the operations that the subject can perform. Each transaction is associated with a single object and each object is associated with one or more transactions. The function *object_transaction* gives the set of transactions that are associated with an object and the function *transaction_permission* gives the permissions available to a transaction.

We summarize the above in the following definition.

Definition 1: The Components of the OBAC Model.

- O, S, and OP (objects, subjects, and operations, respectively).
- $P = 2^{(S \times OP)}$, the set of permissions.
- $PA \subseteq P \times O$, a many-to-many permission to object assignment relation.
- $assigned\_permission(o : O) \rightarrow 2^P$, the mapping of object o onto a set of permissions. Formally: $assigned\_permission(o) = \{p \mid (p, o) \in PA\}$.
- T, the set of transactions.
- $transaction\_object(t : T) \rightarrow O$, the mapping of transaction t to a single object.
- $object\_transaction(o : O) \rightarrow 2^T$, the mapping of object o onto a set of transactions.
- $transaction\_permission(t : T) \rightarrow 2^P$, the mapping of transaction t onto a set of permissions. Formally: $transaction\_permission(t) \subseteq \{p \mid p \in assigned\_permission(transaction\_object(t))\}$.
- $avail\_transaction\_permission(t : T) \rightarrow P$, the mapping of transaction t to a single permission.

Definition 2: Permission Priority.

- $\forall t \in T, \exists p, p' \in P, p \, precedes \, p'$,
$p, p' \in transaction\_permission(t)$
$\Rightarrow avail\_transaction\_permission(t) = p$

Definition 3: Permission Inheritance.

Typically, a single permission is represented as an ACE and a set of permissions are organized in the form of ACL. Permission inheritance occurs between a parent object and its child objects. The INHERITED_FLAG flag of an ACL on the parent specifies that the ACL can be inherited by the child objects or not, however the INHERIT_FLAG

flag of an ACL on a child object specifies whether the child object inherits the ACL of its parent or not. We now define inheritance relationships between permissions.

Definition 3.1: Static ACL Inheritance.

- $\forall o, o' \in O$, $\forall ACLs\ A\ on\ o'$, $o' \in ancestors(o)$

$\wedge\ A.INHERITED\_FLAG = TRUE$

$\Rightarrow assigned\_permission(o) = \{p \mid p \in P,\ p\ in\ A\}$

Definition 3.2: Dynamic ACL Inheritance.

- $\forall o, o' \in O$, $\forall ACLs\ A\ on\ o$, $A'\ on\ o'$,

$o' \in ancestors(o)$,

$\wedge\ A.INHERIT\_FLAG = TRUE$

$\wedge\ A'.INHERITED\_FLAG = TRUE$

$\Rightarrow assigned\_permission(o) = \{p \mid p \in P,$

$p\ in\ A \vee p\ in\ A'\}$

Appendix B. Comparison of PKI, IBE and CPK

B.1. Authentication Based on PKI

The public key infrastructure (PKI) technology has been available for approximately 30 years and appears to meet the requirements for confidentiality and integrity of data, user identification, authentication, and non-repudiation. With PKI, one can maintain her keys and certificates in security and conveniently encrypt data and sign messages. Applications based on PKI include secure web browser, secure e-mail, e-business, e-government, e-banking, and so on. PKI has been imagined to be a magic security elixir, where you can just add a drop to your system and it will become secure.

However, there are some counterviews to PKI [32]. Ellison and Scheier [33] consider that the effect of PKI is overtouted, though it can be used as a solution to many security problems. Rivest [34] raises doubts about certificate revocation lists (CRLs), one common approach to revoking certificates. Clarke [35] indicates that the conventional PKI, built around ISO standard X.509 [36], is inherently ineffectual and privacy-invasive. Sha and Bai [37] discuss the deficiencies of PKI and the shortcoming of the current standard based on certificate, such as X.509, PGP [38] and SPKI/SDSI [39]–[41].

In sum, PKI has several inherent deficiencies and its abilities have been overtouted for a long time. In PKI systems, a public key is bound to the key-holder's name by a certificate authority (CA). But the trustworthiness of CA has long been considered illogical. Many current PKI implementations employ a hierarchical CA model where, since a single CA can service only a limited number of users, PKI increases the scale of key management by adding CAs, which in turn gives rise to the issue of agency expansion and increasing network traffic. Most PKI systems revoke certificates by issuing CRLs. But this is not an assured, secure service. In order to confirm the validity of a certificate, one has to rummage all over the Internet to see if the certificate that she accepted is still OK. In addition, as private digital signature keys attract more attention; more attacks will be directed to these keys. So far there are still very few products available to enable users to safeguard their private keys. It is obvious that more research should be conducted on alternative technologies to PKI.

B.2. Authentication Based on IBE

In 1984 Adi Shamir [42] first proposed the idea of an identity-based cryptosystem in which the public key can be an arbitrary string. However, the fist practical identity encryption scheme was not introduced until 2001 by Boneh and Franklin [43]. Identity-based encryption (IBE) schemes employ the identification of a user's identity as a public key or derive the public key from the user's identity. As a result, "Identity-based encryption schemes enable any pair of users to communicate securely and to verify each other's signatures without exchanging private or public keys, without keeping key directories, and without using the services of a third party" (Shamir, 1984). In contrast to PKI, IBE can work without the support of a CA hierarchy, thereby eliminating the increasing demands on bandwidth incurred by constantly expanded CA facilities. Because of these inherent advantages, IBE is considered a potential alternative technology to PKI.

B.3. Authentication Based on CPK

Although IBE addresses the crisis of trust that confounds PKI and can work without the support of a CA hierarchy, IBE systems still rely on a large number of public parameters to define their operations, and a user of an IBE system needs to obtain these public parameters before any IBE operation can be carried out. So IBE systems can not work without an on-line public parameter server (PPS), which provides IBE public parameters and policy information for an IBE private key generator (PKG). As a result, IBE is not a true two-party authentication process (i.e., devoid of a third party such as CA in PKI).

Nan [29] presents a combined public key (CPK) algorithm, one of the IBE variants, that does not require a third-party CA hierarchy. But unlike IBE, CPK holds only a small number of public parameters, which can be stored in a tiny chip. Thus CPK will be able to obtain the necessary public parameters for a public key from the chip without an on-line PPS. As a result, CPK fully implements peer-to-peer authentication.

The main idea behind CPK is to use a small amount of seeds to produce an almost limitless amount of keys in order to meet the almost limitless demand for keys. The CPK algorithm is based on the discrete logarithm

Table 3. A characteristic comparison of PKI, IBE and CPK systems.

System	*Certificate*	*CA Support*	*Key Management*				
			Generation	*Storage*	*Scale*	*Revocation*	*Protection*
PKI	CA Cert	Yes	Decentralized	On-line certificate database	10^3	CRL	Media, password
IBE	ID Cert	No	Centralized	On-line public parameter server	10^3	Identity	Media, password
CPK	ID Cert	No	Centralized	Build-in chip	10^{48}	Identity	Media, password, active parameter

problem (DLP). It constructs public and private key matrices according to the hardness of DLP, and maps the identity of an entity onto the sequence of the row and column coordinate in these matrices. According to the sequence, the CPK algorithm picks out and combines the matrix elements to generate an enormous number of public/private key pairs. For example, a 32×32 matrix with 192-bit keys that occupies 24KB of memory can generate $32^{32} = 10^{48}$ keys. Thus the size of a public key matrix is small enough to be stored in any accessible media, such as ATM and POS machine, or even be distributed to a user in a smart card. Moreover, CPK first introduces agent and active parameter technologies into key protection in order to withstand the colluded attacks. Besides the protection of traditional physical media and password, the user's private key in a chip is also under the protection of the active parameters.

B.4. Characteristic Comparison of PKI, IBE and CPK systems

Table 3 summarizes the main characteristics and performance properties of key management provided by PKI, IBE and CPK systems. In the table, the *Certificate* column describes the type of certificate used by these authentication systems. CA Cert and ID Cert stand for the authority certificates issued by a certificate authority (CA) and the identity certificates issued by a trusted authority (TA) respectively. The *CA Support* column indicates whether the authentication process needs a third party CA support. The *Key Management* column includes five sub-columns: *Generation*, *Storage*, *Scale*, *Revocation*, and *Protection*, which stand for key generation mode, public key storage, the size that a CA or TA can service, key revocation method, and private key protection, respectively. PKI systems revoke any public key certificate by issuing CRLs, while IBE and CPK systems directly revoke a user's identity. It can be seen that CPK addresses the problems that PKI and IBE can not solve.

Appendix C. Cryptographic Overhead

We tested the raw speed of the cryptographic algorithms used by our system, including the IDEAS and CapSec implementations. These algorithms include traditional symmetrical and non-symmetrical cryptographic

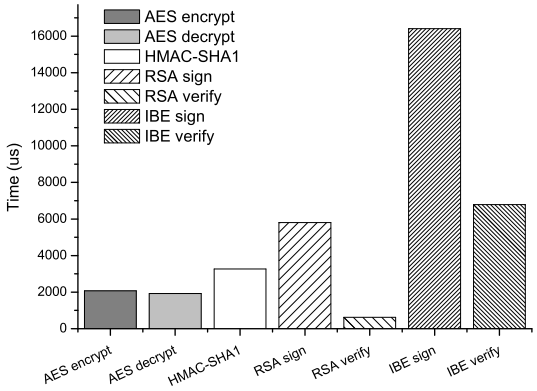


Figure 8. Performance of cryptographic algorithms. Block size is 64KB except for sign & verify, which are done on 160 bits input.

Table 4. Raw Speed of Cryptographic Operations.

	RSA	*IBE*
Hash	-	$11\mu s$
Scalar multiplication	-	$7794\mu s$
Exponentiation	$4675\mu s$	$795\mu s$
paring	-	$6781\mu s$

algorithms as well as up-to-date identity-based cryptographic algorithms; this provided insight into how fast the identity-based algorithms are likely to be when compared to the traditional cryptographic algorithms and how much overhead can be caused when these algorithms are applied to our system.

We used the Hess's identity based signature scheme [44] for the performance evaluation of identity-based signature algorithms, which is implemented using the PBC library [45], and used the OpenSSL crypto library for the AES, HMAC-SHA1 and RSA cryptographic algorithm implementations. The AES and HMAC-SHA1 keys are 128 and 160 bits length respectively, while the RSA and IBE keys are 1024 and 160 bits length respectively. The performance of these cryptographic algorithms is summarized in Figure 8. As the figure shows, the AES and HMAC-SHA1 algorithms with an input of 64KB requires about $2.0ms$ and $3.3ms$ respectively, while with an input of 512 bits, roughly the length of a command, the AES and HMAC-SHA1 algorithms, which provide the confidentiality and integrity protection to commands in the IDEAS implementation, incur latencies of only $17\mu s$ and $14\mu s$ respectively. Such overheads are so sufficiently

insignificant that they should not constitute the primary security cost.

The most expensive operation by far is signature generation, which takes about $5.8ms$ and $16.4ms$ in the RSA and IBE cryptographies respectively. Compared to signature generation, signature verification is cheaper, which costs about $0.6ms$ and $6.8ms$ in the RSA and IBE cryptographies respectively. For further analysis, we tested the raw speed of cryptographic operations in RSA and IBE signature algorithms. As shown in Table 4, the scalar multiplication, exponentiation and paring computation are computationally expensive operations. However, for the RSA algorithm only one exponentiation is required in the signing and verifying steps respectively. Without any precomputation[3], the signing operation in the IBE algorithm require one exponentiation, one hash function evaluation, one paring computation and two scalar multiplication, while the verifying operation require one exponentiation, one hash function evaluation and two paring computation. With precomputation, the signing and verifying operations can be optimized; thus the former requires one exponentiation, one hash function evaluation and two scalar multiplication, while the latter requires one paring computation and one hash function evaluation. It should be noted that though the signature takes the most amount of running time, it usually occurs only in the logging process.

Acknowledgments

The authors would like to thank Dongliang Lei, Wei Yan, Junjian Chen, Qinhua Yan, Peng Li, Anli Chen and other members of National Storage System Laboratory at the Huazhong University of Science and Technology for their hard work and help in implementing the IDEAS system. This work was supported by National Basic Research Program of China (973 Program) under Grant No.2004CB318201, National Science Foundation of China under Grant No.60703046 and No.60873028, US National Science Foundation (NSF) under Grant No. CCF-0621526, the Program for New Century Excellent Talents in University NCET-04-0693 and NCET-06-0650, the Program for Changjiang Scholars and Innovative Research Team in University under Grant No. IRT-0725 and Wuhan Project 200750730307.

References

[1] P. J. Braam, "The Lustre storage architecture," *http://www.lustre.org/documentation.html, Cluster File Systems, Inc.*, Aug. 2004.

[2] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proc. of SOSP'03*, Oct. 2003.

3. See Section 2 in [44].

[3] G. A. Gibson *et al.*, "A cost-effective, high-bandwidth storage architecture," in *Proc. of 8th ASPLOS*, Oct. 1998.

[4] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas activescale storage cluster-delivering scalable high bandwidth storage," in *Proc. of 8th ASPLOS*, Nov. 2004.

[5] O. Rodeh and A. Teperman, "A scalable distributed file system using object disks," in *Proc. of Mass Storage Systems and Technologies Conf.*, 2003.

[6] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proc. of FAST'02*, Jan. 2002.

[7] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. of OSDI '06*, 2006.

[8] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, and D. D. E. Long, "File system workload analysis for large scale scientific computing applications," in *Proc. of the Conference on Mass Storage Systems and Technologies*, Apr. 2004.

[9] M. K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D. Andersen, M. Burrows, T. Mann, and C. A. Thekkath, "Block-level security for network-attached disks," in *Proc. of FAST '03*, 2003.

[10] A. Azagury, R. Canetti, M. Factor, S. Halevi, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, and J. Satran, "A two layered approach for securing an object store network," in *Proc of IEEE Security in Storage Workshop*, 2002.

[11] M. Factor, D. Nagle, D. Naor, E. Riedel, and J. Satran, "The OSD security protocol," in *Proc. of 3rd IEEE Security in Storage Workshop*, 2005.

[12] Z. Niu, K. Zhou, D. Feng, H. Jiang, F. Wang, H. Chai, W. Xiao, and C. Li, "Implementing and evaluating security controls for an object-based storage system," in *Proc. of MSST'07*, Sep. 2007.

[13] H. Gobioff, "Security for a high performance commodity storage subsystem," Ph.D. dissertation, Carnegie Mellon University, July 1999.

[14] R. A. Oldfield, A. B. Maccabe, S. Arunagiri, T. Kordenbrock, R. Riesen, L. Ward, and P. Widener, "Lightweight I/O for scientific applications," Sandia National Lab, Tech. Rep. 2006-3057, May 2006.

[15] Y. Zhu and Y. Hu, "Snare: A strong security scheme for network-attached storage," in *Proc. of the 22nd Symp. on Reliable Distributed Systems*, 2003.

[16] C. A. Olson and E. L. Miller, "Secure capabilities for a petabyte-scale object-based distributed file system," in *Proc. of the 1st ACM Workshop on Storage Security and Survivability*, Nov. 2005.

[17] A. W. Leung and E. L. Miller, "Scalable security for large, high performance storage systems," in *Proc. of the 2nd Workshop on Storage Security and Survivability*, 2006.

[18] A. W. Leung, E. L. Miller, and S. Jones, "Scalable security for petascale parallel file systems," in *Proc. of SC07*, Nov. 2007.

[19] B. C. Reed, E. G. Chron, R. C. Burns, and D. D. E. Long, "Authenticating network-attached storage," in *Proc. of Hot Interconnects VII*, Aug. 1999.

[20] V. Kher and Y. Kim, "Decentralized authentication mechanisms for object-based storage devices," in *Proc. of the Second IEEE International Security In Storage Workshop*, 2003.

[21] B. C. Neumann, J. G. Steiner, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Proc. of Winter USENIX Conference*, 1988.

[22] *SCSI Object-Based Storage Device Commands -2 (OSD-2)*, Project t10/1729-d, revision 3 ed., T10 Technical Committee, NCITS, January 2008.

[23] *Draft Standard for Information Technology - Portable Operating System Interface (POSIX) - Amendment: Protection, Audit, and Control Interfaces*, Portable Applications Standards Committee (PASC), November 1992.

[24] M. M. Swift, A. Hopkins, P. Brundrett, C. V. Dyke, P. Garg, S. Chan, M. Goertzel, and G. Jensenworth, "Improving the granularity of access control for Windows 2000," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 4, pp. 398 – 437, Nov. 2002.

[25] Y. Yang, R. Ding, and Y. Min, "Object-based access control model," *Automation of Electric Power Systems*, vol. 27, no. 7, pp. 36 – 40, 2003, (in Chinese).

[26] C. Li, C. Liu, M. Hong, and W. Cai, "Object-based multi-subject access control model," *Computer Integrated Manufacturing Systems*, vol. 11, no. 3, pp. 342 – 346, 2005, (in Chinese).

[27] "Trusted Computer Security Evaluation Criteria," DOD 5200.28-STD, Department of Defense, 1985.

[28] J. F. Barkley, A. V. Cincotta, D. F. Ferraiolo, S. Gavrila, and D. R. Kuhn, "Role based access control for the world wide web," in *Proc. 20th NIST-NCSC National Information Systems Security Conference*, 1997, pp. 331–340.

[29] X. Nan, *Identity Authentication Based on CPK*, 1st ed. Beijing, China: National Defense Industry Press, January 2006, (in Chinese).

[30] Q. Yuan and S. Li, "A new efficient ID-Based authenticated key agreement protocol," *Cryptography ePrint Archive, Report 2005/309*, March 2005.

[31] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology-Crypto'87*, 1987, pp. 369–378.

[32] T. Moreau, "Thirteen reasons to say 'no' to pubilic key cryptography," CONNOTECH Experts-conseils, Inc., http://www.connotech.com/13REAS.HTM, Draft paper, March 1998.

[33] C. Ellison and B. Schneier, "Ten risks of PKI: what you're not being told about public key infrastructure," *Computer Security Journal*, vol. 16, no. 1, pp. 1–7, Winter 2000.

[34] R. L. Rivest, "Can we eliminate certificate revocations lists?" *Lecture Notes in Computer Science*, vol. 1465, p. 178, 1998.

[35] R. Clarke, "Conventional public key infrastructure: An artefact ill-fitted to the needs of the information society," November 2000, http://www.anu.edu.au/people/Roger.Clarke/II/PKIMisFit.html.

[36] R.Housley, W.Ford, W.Polk, and D.Solo, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, RFC 2459, January 1999.

[37] Y. Sha and S. Bai, "On the research and analysis of the main problem of PKI," *Microelectronics & Computer*, no. 6, pp. 18–21, 2002, (in Chinese).

[38] S. Garfinkel, *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1995.

[39] C. Ellison, *SPKI Requirements*, RFC 2692, IETF, September 1999.

[40] C. Ellison, B. Frantz, R. Rivest, B. Thomas, and T. Ylonen, *SPKI Certificate Theory*, RFC 2693, IETF, September 1999.

[41] R. L. Rivest and B. Lampson, "SDSI-a simple distributed security infrastructure," September 1996, http://people.csail.mit.edu/rivest/sdsi10.html.

[42] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proc. of CRYPTO 84 on Advances in cryptology*, 1985.

[43] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," *Lecture Notes in Computer Science*, 2001.

[44] F. Hess, "Efficient identity based signature schemes based on pairings," *SAC 2002, LNCS 2595, pp. 310C324, 2003.*

[45] B. Lynn, *PBC Library Version 0.4.12*, http://crypto.stanford.edu/pbc/.