

Group-based Security in a Federated File System

Max Berger
Texas Tech University
max@berger.name

Michael Sobolewski
Texas Tech University
sobol@cs.ttu.edu

Abstract—The SILENUS federated file system was developed by the SORCER research group at Texas Tech University. The distributed file system with its dynamic nature does not require any configuration by the end users and system administrators.

Managing security in a metacomputing system is a new challenge. It must be ensured that every user has a valid authentication and authorization to view, modify, and create files in the system spread across many heterogeneous computers that to individual requestor, it looks and acts like a single computer. User management is a must be on a metacomputing system and scale well. Existing user credential databases must be incorporated as secure data services if present.

In this paper a new scalable authentication model for federated file systems is described. In this model users authenticate to an authentication service for their identity and a group manager service for their collaborative groups membership. The group manager service provides an authorization token that can be used to invoke service-oriented functionality of the federated file system. The group manager service uses existing user credential databases as its back-end. There may be any number of group manager services on the network with different user administration domains to provide desirable scalability. Multiple replicated group manager services for the same user base can provide for increased reliability. A scaled-down replica called nomadic group manager service provides support for disconnected operations. It contains the necessary credentials for a single user to use the system while being disconnected from the main network.

I. INTRODUCTION

Under the sponsorship of the National Institute for Standards and Technology (NIST) the Federated Intelligent Product Environment (FIPER) ([1], [2], [3]) was developed (1999-2003) as one of the first service-to-service (S2S) grid computing environments. The Service-Oriented Computing Environment (SORCER) ([4], [5]) builds on the top of FIPER to introduce a federated metacomputing environment with all basic services necessary to support service-oriented programming. It provides an integrated solution for metacomputing systems.

Building on the OO paradigm is the service-object oriented (SOO) paradigm, in which the objects are distributed, or more precisely they are remote (network) objects and play some predefined roles. A service provider is an object that accepts remote messages, called exertions, from service requestors to execute an elementary item of work (statement) – a service task, or a composite item of work (procedure) – a service job.

Any exertion becomes a SOO program that is dynamically bound to all relevant and currently available service providers on the network. This collection of providers dynamically participating in this federated remote invocation is called an exertion federation. This federation is also called a virtual metacomputer as federating services are located on multiple

physical compute nodes held together by a SOO infrastructure so that, to the individual requestor, it looks and acts like a single computer.

The SORCER environment provides the means to create interactive SOO programs and execute them without writing a line of source code. Service jobs and tasks can be created using interactive user interfaces downloaded directly from service providers. Using these interfaces the user can execute and monitor the execution of exertions in the SOO metacomputer. The exertions can be persisted for later reuse. This feature allows the user quickly to create new applications or programs on the fly in terms of existing exertions.

The SILENUS federated file system was designed and developed to provide data access for SOO programs. The SILENUS system itself is a collection of service providers that use the SORCER framework for communication. In this paper a relevant security framework is described to allow the exertion federation for a secure collaborative data access.

Since services are connected dynamically in an S2S environment, service security has become more difficult to implement and maintain than in a static computer network. In addition, new levels of service access (directly through interfaces and indirectly via common service(Exertion):Exertion invocations, by any faceless service peers) are providing new opportunities for unauthorized interaction and security breaches. A good security framework will have to address the following attributes: [6] [7]

- identification and authentication,
- authorization,
- resource control and containment,
- confidentiality and integrity,
- non-repudiation, and
- auditing.

In addition to the security framework, the following security measures are also recommended:

- educate users about service security concerns and policies;
- implement a break-in detection plan to detail when to look at audit information and specify what an auditor provider would look for;
- implement a recovery plan detailing how to recover from a break-in.

SORCER originally provided a simple File Store Service (FSS). It supports filtering out information from remote files, thus reducing the amount of data transfers between providers. However, it is provided as a service with a centralized database and as such not a true metacomputing application. [8]

To improve reliability and performance, file replication services were added to SORCER. These services allowed for the replication of file data on different nodes in the data grid. This greatly reduced data access time. [9]

SILENUS completes the step from a traditional client-server file-system to a network-centric system. Instead of storing data on one particular node or in a particular service, it is the federation of several federated services that provide the file system. Data is no longer stored in a single service. It is split up into different services for file content, file metadata, and management data. SILENUS provides a true data grid solution to complete SORCER's metacomputing grid. [10] [11]

This service oriented file system needs at least the following security functionality: identification, authentication, authorization, and confidentiality. Users need to be identified with a name or account. They need to be authenticated through a password or a unique artifact, such as a smart card or fingerprint. The authorization will then determine what types of activities are permitted. Data stored in the SILENUS system must be kept private while being transmitted over an open network and stored on insecure nodes.

Authentication is the process of determining the authenticity of a message or user. It can be used to verify the identity of a user, service requestor and provider, or that a message has not been tampered with. Authentication can be implemented using different approaches, in particular: message digests, message authentication codes (MACs), and digital signatures. The latter approach is very valuable to SOO metacomputing as it provides both a guarantee of the source of the data and proof that the data has not been tampered with. This approach allows the efficient use of digital certificates. A digital certificate is essentially a signed statement by the X party that the Y party's public key belongs in fact to Y. The Public Key Infrastructure (PKI) and its simpler version (SPKI) are essentially systems for managing public-key cryptography used for the proposed security in the SILENUS file system. In particular service requestors and providers are identified by X.509 digital certificates. While these certificates are usually called "public key certificates", this paper also uses the term "identity certificate" or "identity" for short to emphasize their use to securely identify an entity in a metacomputing system.

The classic security concept is for the client node to authenticate directly to the node that provides the service. This approach works well in a traditional client/server environment. However, in a large distributed environment this would require the user credentials to be replicated among all service providers. Obviously this is an administrative challenge. It is also not very secure, as credentials may be intercepted or read by local administrators and users in the network.

A better approach uses tokens issued by an authentication service. Instead of authenticating directly with the service provider, a user will authenticate with one central server - a key distribution center (KDC). This server holds the keys for all the users and services in the network. The server will then issue a token to the user. This token can be used to authenticate with services providers, which will verify the token authenticity

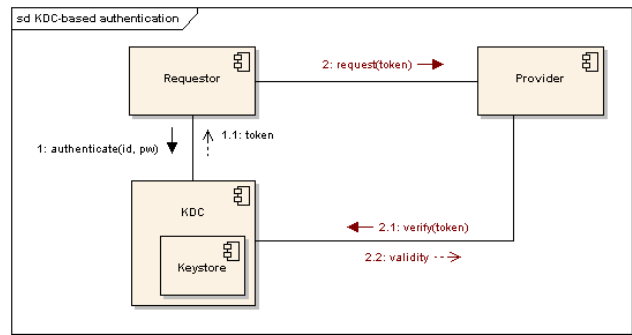


Fig. 1. Security model with key distribution center (KDC). A KDC is a trusted service that knows the keys for all the nodes. If a new service is deployed on the network, this service and the KDC need to be configured with a key for the new node.

with the issuing authentication server. This approach is used by Kerberos [12]. It works very well for smaller distributed applications. In this solution, the issuing server has to be accessed from all participating services. It will therefore not scale well. Requiring a central server to be accessible creates problems with organizational firewalls, which restrict the use of this model. Figure 1 illustrates this key distribution center case.

A problem with most existing security concepts is that they do not allow existing authentication and user databases to be re-used. Every system has its own user and password database. Most systems can import users from other systems, but importing passwords is very often a problem. Passwords are usually stored in some specific encrypted formats and cannot be used across other authentication system. The presented solution allows adapting applications to different credential providers.

Special credential mechanisms such as fingerprint scanners and smart cards are hardly ever supported. In a few cases, some applications such as a user login are adapted for these devices. However, the keys stored on such a system could be used for all kinds of services. Smart cards have been used successfully to authenticate users with services in SORCER. [13]

Existing systems, such as TLS [14], are targeted towards the direct communication between two partners. In a federated system, however, requests may be sent through intermediate services instead of being sent directly. Most existing remote file systems are based on such end-to-end authentication and encryption mechanisms.

The more advanced distributed file systems AFS [15] and Coda [16] are based on the Kerberos authentication mechanisms. Kerberos is currently the most widely used solution for authentication in distributed applications, but has some of the shortcomings described above.

What is needed is a scalable and flexible security system for the metacomputer that makes use of existing credentials. It should support different administrative domains, but still provide one but flexible unique privacy and authentication mechanism.

In this paper we describe such an approach for the SORCER

environment in general and for the SILENUS file system in particular. First, requirements for federated computing are identified. Then, existing authentication and identification mechanisms are investigated. Next, a new service called “Authentication Adapter” is introduced. A novel service for managing user groups, called “Group Management Service” is also presented. Then, a service called “Nomadic Group Management Service” is introduced to support disconnected operations. All identification and group management services can be discovered dynamically and clients can federate with them transparently with no a priori knowledge about their location. The collaborative protocol for such authentication-authorization federation is investigated. The presented group management system is scalable and reliable as needed dynamically federating identification and group management services can be provisioned autonomically [?].

II. REQUIREMENTS

To provide the scalable and flexible security system the following two assumption are made:

- 1) In a large-scale system it is more important to recognize returning users than to recognize specific users. It is not important which identity a user has as long as the user has the same identity when connecting again. Using this assumption a user could provide own credentials. As long as it is ensured that the credentials are kept secure, the user can always be uniquely identified.
- 2) In a metacomputing system all sharing is from one user to another. The user becomes an administrator when sharing files with another one. Thus, user identities and group identities have to be simple and uniform not around administrative domains but around users.

III. BACKGROUND

Allowing the user to provide own credentials can lead to an explosion of user accounts or certificates. Therefore a user account has to be verifiable by a trusted source. This is commonly referred to as a trusted third party model.

In a trusted third party model a user authenticates with an authentication service. The authentication service will then provide verification that a given user is who she or he claims to be. The service provider can then verify that the user is authenticated by this authentication service. The authentication services itself are certified by a certificate authority (CA).

The list of third parties should be small and change seldom. This information will have to be configured on every service provider. It should change as little as possible. Every change would require additional administration.

A trusted third party can be any service that provides a user base. It could be an LDAP server, a Windows domain server, a Kerberos server, or a trusted party signing public keys. It is only required that the server can verify users. In SORCER this is implemented with X.509 certificates using keystore and truststore services. [17] These certificates are also used with TLS (SSL) security protocol to make a secure authenticated

connection between two parties: service requestor and service provider.

A. Asymmetric cryptography

Asymmetric cryptography uses a pair of matching private and public keys. The private key can be used to sign a message. Knowing the public key, another entity can verify that a certain message was indeed signed using the matching private key. The authenticity of a public key can be verified through configuration and such a key is called a trusted key. This allows a private and public key-pair to be used to define unique identities in the network.

To be secure in a network environment, the private key must never leave the node it is stored on. All messages and identities to be signed must be sent to the network node containing the private key and signed locally. The identity of the user or host requesting the digital signature must be verified before creating the signature.

B. Public Key Infrastructure (PKI)

The basic trusted third party model requires the service provider to be able to talk to the authentication service directly. This is undesirable, and very often not possible. It also does not define how the credentials are passed to and verified by the service provider.

A standard for credentials needs to be defined. That standard should be common, and should allow verification without talking back to the original service. One of the commonly used standards is provided by an X.509 public key certificate. Such a certificate is defined as the public key of a user, together with some other information signed by a third party’s private key. That third party is known as the certificate authority (CA). Public Key infrastructure (PKI) is essentially a system for managing public-key cryptography. [18] [19] PKI is an attempt to integrate a number of protocols and standards into a more unified system that provides secure services.

User credentials can be any type of unique user identification and related information. The most common authentication uses a username and a secret password. This type of authentication requires no special hardware on the user’s host. If special hardware is present, a more sophisticated mechanism can be used.

Usually PKI authentication is done by a service requestor using its private key to perform a cryptographic operation on a nonce the service provider supplies, and then transmitting the result to the service provider. The provider checks the result using the requestor’s public key. In that case a private key has to be persisted by the requestor. In the presented approach having permanent private keys stored by the user is avoided and delegated to the authentication service. We assume that the requestor can create a temporary private/public key pair if needed, for example to establish SSL secure communication channel. In that case the requestor’s public key can be signed by the authentication service that will authenticate the user with his username / password.

In PKI there are usually multiple certificate authorities. A global CA is used to sign the identity of other lower-level CAs. In our approach, each authentication service is a sub-CA signing user keys thus creating a certificate chaining. The signature on the user key can then be used to verify that it was signed by a certain authentication service. The identity of the authentication service can be verified by checking if its certificate is signed by the global CA. Key management is reduced since on the provider side, only the identity of the global CA needs to be configured. A private key for each authentication service must be created and signed by the global CA. There are usually less authentication services than requestors, which also reduces the key management overhead.

Thus the authentication provider has to satisfy the following two requirements:

- 1) The provider has to provide a public key, which is certified by a trusted global CA.
- 2) The provider also has to be able to sign small messages, such as public keys providing an identity.

The actual private key will never have to leave the authentication service. A service provider accessed by a requestor can then verify the requestor's public key identity with its own trust-store containing a public key of the global CA. Figure 2 illustrates authentication with the global CA and the described authentication service.

The PKI provides support for identities. An identity consists of unique service identification and a user or entity identification that is unique on this service. The service can guarantee that it knows the user under this name. Each service in the SORCER network has a unique provider-id. This id is used for the identification of the service. Group and entity ids are text strings, such as "admin".

C. Simple Public Key Infrastructure (SPKI)

The purpose of the Simple Public Infrastructure (SPKI) is to communicate permissions from a keyholder to another. SPKI's primary objective is to provide a service provider with the evidence that the holder of a public key is ultimately authorized for a request signed by its matching private key. This approach contrasts with PKI efforts that attempt to bind keys to identities, and leave authorization to be handled by mapping requestor's identity to authorization. Thus, using PKI, if you know a name for a service requestor you know its identity, then you might know whether it is authorized to do or have something they request. This assumption is true in small SOO environments. That world no longer exists in environments SORCER is designed for. Any certificate mechanisms based on global names (e.g., X.509) fail to scale well. In the Simple Distributed Security Infrastructure (SDSI), an identifier is valid only locally to the service requestor who creates it but the underlying raw public key is valid globally. In SPKI, an authorization grant is made only locally. If the authorization grant is needed to someone beyond a given locality, then that grant should be delegated through a chain of local relationships. [20] [21] [22]

SPKI has two types of certificates: name certificates, which define local names, and authorization certificates, which confer authorization on a key or a name. SPKI name certificates are comparable to X.509 and are used for example as identity certificates for users, services, and groups.

In SPKI, a service requestor creates its own local identity consisting of a private and public key. The public key is then sent to an authentication service, along with the username and password. The authentication service verifies the password and then signs the public key. The signed key is then sent back to the requestor. When a request is made, the request is signed with the local identity. This local identity is passed along with the request and other credentials. The provider can then examine the local identity to verify that the local identity is signed by the authentication service. The provider can chain the verification and verify that the authentication service is vouched by the global CA, which it trusts, which then makes a local identity trusted. Figure 3 illustrates the SPKI-based authentication process. Figure 4 shows the keys, certificates and configuration on the involved services.

IV. AUTHENTICATION ADAPTER

To provide support for existing user databases that are not based on X.509 an authentication adapter service is needed. This adapter service provides the required services and uses the existing authentication service as its back-end. It will have to be run on a secure system.

Unlike existing authentication services, such as Kerberos, these adapters allow to use any user base as a backend. It is not required to keep multiple different accounts for the same user.

The adapter service checks the user credentials against an existing user database. The user database may be a Unix passwd file, an NT Domain, or any other system that can authorize users with given credentials. If the credentials match the ones in the user database, then the identity of the user is assumed to be correct. His or her local SPKI identification will be signed.

There may be multiple authentication adapters to provide for scalability. Each authentication adapter should authenticate against a specific user database backend. The authentication backend identifier is part of the user id since a user with the same username on different authentication systems may or may not be the same user. The drawback of this solution is that a user that has accounts on multiple systems will also have multiple accounts in the SORCER system.

It is important that the adapter service is able to create new keys for users that have not yet authenticated themselves with this service. If a new user authenticates him/herself, a new pair of private/public key must be created. The public key must be automatically signed by the adapter. The adapter certificate is listed as a trusted third party public key in the SORCER truststores. Figure 5 gives an example of the authentication process using an authentication adapter service.

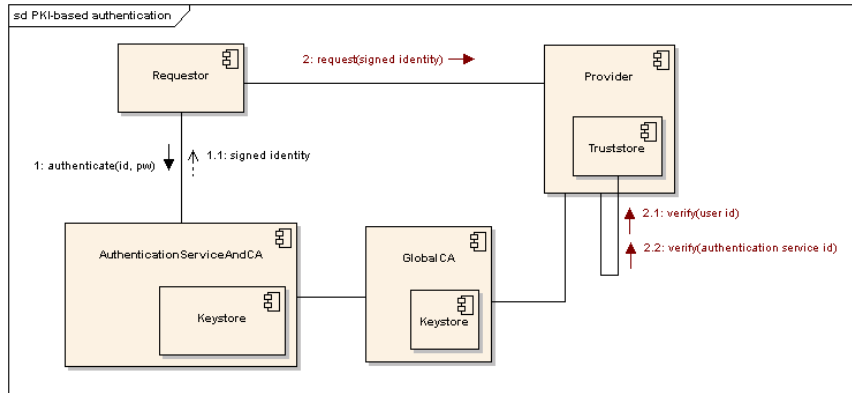


Fig. 2. Authentication with PKI certificates. Each service provider is responsible for knowing its own private key. All the public keys are accessible from one common truststore.

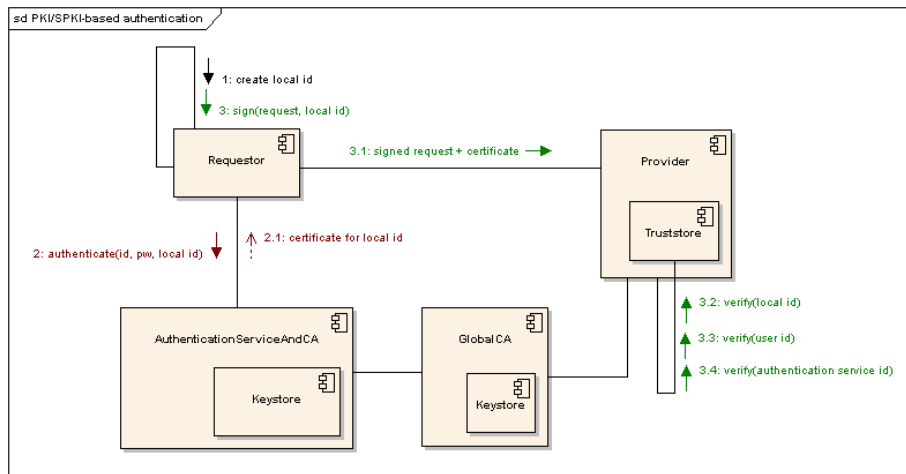


Fig. 3. Authentication with self generated identification (SPKI name certificates) signed using an authentication service using PKI. The service requestor generates a temporary identity consisting of a private and public key. The public key is then sent to the authentication service, along with credentials. The authentication service provides a certificate stating that the tempid belongs to the user.

Private Key	A (Local Id)
Certificate	[max@1234 has key A] _B

(a) Requestor

Provider-Id	1234
Private Key	B
Certificate	[1234 has key B] _C

(b) AuthenticationService

Private Key	C
-------------	---

(c) Global CA

Configuration	C is trusted
---------------	--------------

(d) Provider

Fig. 4. Keys, certificates, and configuration stored on different hosts for SPKI. The certificates here are shown after all authentications occurred.

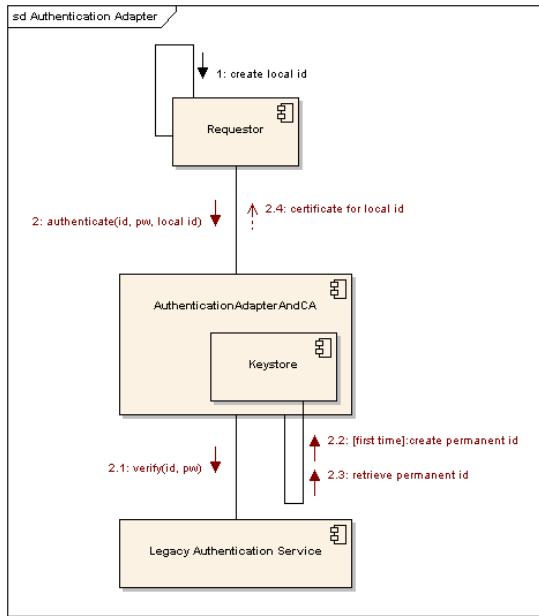


Fig. 5. An authentication adapter service. The adapter connects to a legacy authentication system to verify the existing credentials of requestor. If this is the first time for a requestor to authenticate the adapter creates a new identity in its keystore. It then uses this permanent identity to sign a requestor temporary identity and sends it back to the requestor.

V. USER GROUPS

The system described so far provides support for individual users. It describes the security mechanisms found in existing systems. However, it lacks support for groups of users and different roles they might play. An extra step is needed to support user groups and roles. A group is a set of users that share some common security privileges. A role is a named list or group of privileges that are collected together and granted to users or other roles. Belonging to a group or assuming a role is the same in the context of a file system. If a user belongs to the group of administrators he or she may assume the role of an administrator. To support user groups, a new service provider called “Group Manager Service” (GMS) provider is introduced.

A. Group Manager Service (GMS) Provider

The group manager service (GMS) provider defines a mapping from users to groups. Given a user identity, the GMS provider will supply credentials for all groups this user belongs to. There may be multiple GMS providers managed by different administrators.

The GMS provider manages group identities and creates new group identities per user requests. It is a combination of a group service provider and the authentication service. The process is therefore similar to the presented already authentication with authentication services. A user authenticates her- or himself with an authentication service. This authentication service will verify the user’s identity. Using the identity provided by the authentication service, the user then authenticates with a GMS provider for group membership.

The GMS provider contains a database of group identities with a mapping of users to managed groups. A group name is defined by a proper group name and the unique id of the GMS provider itself. The GMS provider verifies the authenticity of the user and if she or he belongs to the requested group. The GMS provider will then confirm the group membership of the user by creating a signed certificate for the local group identity of the service requestor. The certificate is signed with the private key of the trusted GMS provider.

The mapping from users to their containing groups must be created by an administrator. This might seem to contradict the requirement of a scalable security framework. However, there may be any number of group manager services, which can be managed by different administrators. A local workgroup may therefore run their own GMS with their own groups, so this approach still scales well. Another approach would be to allow every user to create their own groups. This can be limited by granting permissions to subset of users and groups. Allowing every user to create own permissions may lead to manageability issues and will have to be further investigated.

Please note that requestors creates their own identity and group identity. Both identities are certified by a trusted authentication and GMS provider correspondingly. The user can play a role as identified by the user id or group id. While user certificates are unique, there will be multiple corresponding certificates to the same group id. This differs from the original PKI approach, where a certificate for a specific distinguished name (id) is unique. Here the same group name associated with different local group keys still can be related to the same GMS group. In the presented case, a group-based service request is signed with the private group key that users holds only, along with the group membership certificate containing a matching and trusted (vouched by its GMS) public key.

The requestor can obtain certificates for the original user identity and related group identities. Each certificate (public key) is valid for one user id or one group id. The user may select which key to use as a role when requesting a service from a provider. The request will have to be signed with the matching private key and sent to the provider along with the related certificates. The provider can then verify the requestor identity or group membership of the requestor. Figure 6 gives an example of group manager behavior. Figure 7 illustrates the keys, certificates, and configurations for the involved services.

It is important to realize that groups with the same name may exist on multiple GMS providers. Thus, a certificate group name is the combination of the proper group name and the id of the GMS provider that vouches for its managed group. This leads to a synchronization problem in the case of replicated GMS providers. Each replicated GMS provider keeps an updated replica of all the groups defined by its master GMS provider. Thus, all replicated GMS providers of the same type certify the groups with the same GMS id so that the group id looks always identical to any service provider independently what GMS replica is used.

The SILENUS file system currently has support for Unix-like permissions with read, write, and execute bits for users,

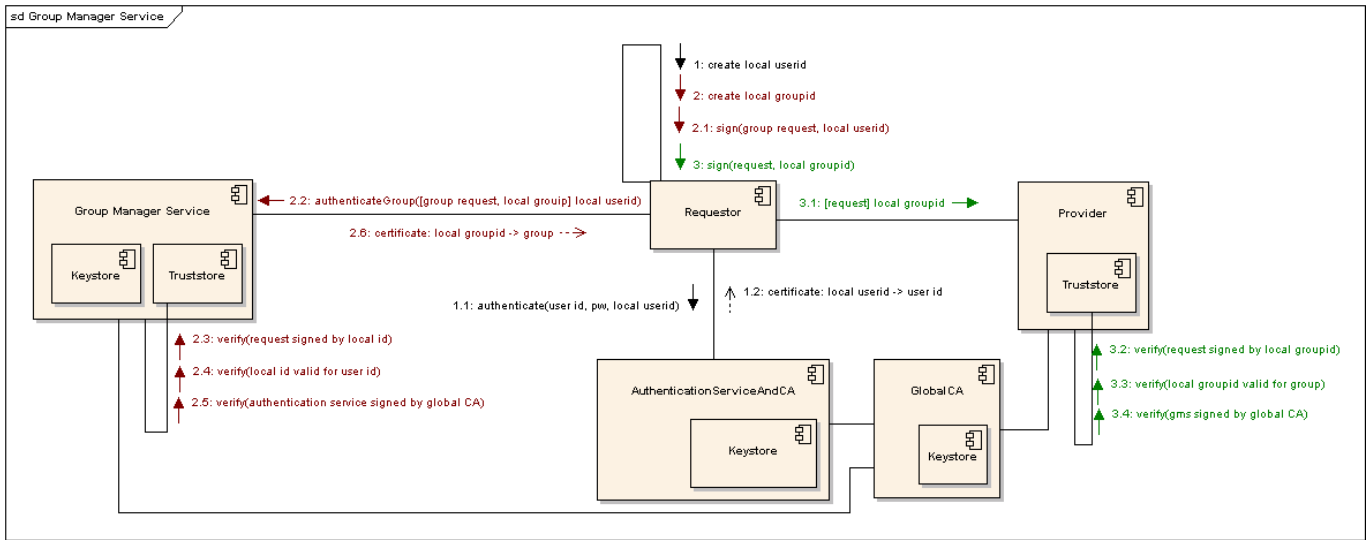


Fig. 6. Authentication via group manager service. The requestor creates a local identity for the user. It then authenticates with an authentication service. The authentication service provides a certificate stating that the local user id is valid for this user. The requestor then creates a local group identity. With the userid certificate the requestor authenticates with a group manager service. The GMS provider can furnish a certificate stating that this local group id is valid for the requested group. The requested group certificate is signed with the private key of the GMS provider. The group (role) certificate can then be used to request a service from any provider enforcing group permissions, for example accessing files in the SILENUS file system.

Private Keys	D (Local UserId) E (Local GroupId)
Certificate	[max@1234 has key D] _B
Certificate	[admin@5678 has key E] _F

(a) Requestor

Provider-Id	1234
Private Key	B
Certificate	[1234 has key B] _C

(b) AuthenticationService

Provider-Id	5678
Private Key	F
Certificate	[5678 has key D] _C
Configuration	max@1234 ∈ admin

(c) GMS provider

Private Key	C
Configuration	C is trusted

(e) Provider

Fig. 7. Keys, certificates, and configurations stored on different hosts for use with a GMS provider. The certificates here are shown after all authentication occurred. The group request to the GMS is signed with the requestor's key D. The request to any service provider is signed with either key D (the userid is selected as a role) or with key E (the groupid is selected as a role).

groups, and everyone else. File access permissions are managed by SILENUS metadata store services. The permissions bits itself are readable for everyone. The SILENUS requestor retrieves the permission bits from the SILENUS file system. It checks if it has the appropriate user identity. If not, it checks which access group a file belongs to. The access group contains the group name and the identity of the GMS provider used. Using this information, the requestor can find and connect to the relevant GMS provider for this particular access group and try to authenticate there. If the user is part of the access group, the GMS provider will provide the group certificate, which the requestor can then use to access a file from the SILENUS file system.

Using multiple and replicated group manager services provides for scalable and reliable administration. A smaller part of a larger organization, such as a department at a university, may provide their own group manager service and its replicas. The credentials from this group manager service can be used to authenticate access to local resources, such as departmental file storage or lab access.

Splitting up of security credentials to be managed by authentication services and group manager services provides support for a modular and flexible solution. Users can have centrally managed accounts, but their privileges may be controlled by individual departments of the organization. Management for departmental groups can be delegated to local administrators

without giving them full access to underlying resources.

B. Nomadic GMS

The group manager service may be replicated to different nodes with subset of its group database. Since the GMS has a copy of its group database, it may only be replicated in to nodes that are trustworthy. In most cases, servers in the locally managed department of organization are trustworthy, and sometimes client hosts, if their users do not have local administration rights. Replicating a GMS gives the usual benefits of replication, such as reliability and scalability.

To support disconnected operation, a subset of the group database existing on a particular remote GMS may be copied in runtime to the user's host as the volatile (no persistent storage). A user may need to use her credentials while not connected to the network, to access data stored on a local nomadic system, such as a laptop. To provide access, a subset of the groups may be copied onto the user's machine. This is supported by the nomadic GMS. The nomadic GMS can replicate only the groups that are relevant to a particular user and in volatile state. The user can then access these credentials locally, providing support for disconnected operation.

VI. CONCLUSION

Security is often very well implemented in small, closed systems. Users have to remember passwords for each and every account they have. The new system proposed in this paper solves group authentication by introducing a scalable service-oriented security system. In this model users authenticate to an authentication service for their identity and a group manager service for their collaborative groups membership.

Providing authentication services by leveraging existing authentication services allows for reuse of existing accounts. Unlike other systems, no migration of user data is necessary. Any existing authentication system may be reused.

The system is scalable and reliable as needed dynamically collaborating identification and group manager services can be provisioned autonomically. There may be any required number of authentication and group manager services. They may provide authentication services for a small or a large network. Decoupling groups from accounts makes the system manageable with a large number of users and groups.

The presented authentication model is also not centralized, so diversifies and eliminates identification and authorizations bottlenecks and potential single-point failures. It is up to the individual service provider which authentication services to accept. Different departments of the same organization may accept different credentials, or they may all accept the same credentials.

The presented security model requires a service provider to keep track of the permissions for individual users and/or group. This is sufficient for a file system such as the SILENUS file system. For a more general service-oriented solution the permission model needs to be provided by a separate authorization service. An authorization service in SORCER is

currently under development that will capture more complex permission structures for a general set of services.

The model described in this document fulfills all requirements for a truly scalable, manageable, distributed security model for federated file systems required in metacomputing environments like SORCER.

REFERENCES

- [1] M. Sobolewski, "Federated P2P services in CE environments," in *Advances in Concurrent Engineering*. A.A. Balkema Publishers, 2002, pp. 13–22.
- [2] —, "FIPER: The federated S2S environment," in *JavaOne, Sun's 2002 Worldwide Java Developer Conference*, San Francisco, 2002, <http://sorcer.cs.ttu.edu/publications/papers/2420.pdf>.
- [3] R. Kolonay and M. Sobolewski, "Grid interactive service-oriented programming environment," in *Concurrent Engineering: The Worldwide Engineering Grid*. Tsinghua Press and Springer Verlag, 2004, pp. 97–102.
- [4] S. Soorianarayanan and M. Sobolewski, "Monitoring federated services in CE," in *Concurrent Engineering: The Worldwide Engineering Grid*. Tsinghua Press and Springer Verlag, 2004, pp. 89–95.
- [5] SORCER, "Laboratory for Service-Oriented Computing Environment," Mar. 2007, <http://sorcer.cs.ttu.edu/>.
- [6] C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World (2nd Edition)*. Prentice Hall PTR, 2002.
- [7] L. Gong, G. Ellison, and M. Dageforde, *Inside Java 2 Platform Security: Architecture, API Design, and Implementation (2nd Edition)*. Prentice Hall PTR, 2003.
- [8] M. Sobolewski, S. Soorianarayanan, and R.-K. Malladi-Venkata, "Service-oriented file sharing," in *CIIT conference (Communications, Internet and Information Technology)*. Scottsdale, AZ: ACTA Press, Nov. 2003, pp. 633–639.
- [9] V. Khurana, M. Berger, and M. Sobolewski, "A federated grid environment with replication services," in *Next Generation Concurrent Engineering*, ISPE. Omnipress, 2005.
- [10] M. Berger and M. Sobolewski, "Silenus - a federated service-oriented approach to distributed file systems," in *Next Generation Concurrent Engineering*, ISPE. Omnipress, 2005.
- [11] M. Berger, "SILENUS - a service oriented approach to distributed file systems," PhD Dissertation, Texas Tech University, Department of Computer Science, Dec. 2006.
- [12] B. C. N. J. G. Steiner and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Winter 1988 USENIX Conference*. Dallas, TX: USENIX Association, 1988, pp. 191–201. [Online]. Available: <http://julmara.ce.chalmers.se/Security/usenix.PS.gz>
- [13] S. Bhatla, "Smart card authentication and authorization framework (SCAF)," Master's thesis, Texas Tech University, Lubbock, TX, May 2005.
- [14] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," RFC 4346 (Proposed Standard), Apr. 2006, updated by RFCs 4366, 4680, 4681. [Online]. Available: <http://www.ietf.org/rfc/rfc4346.txt>
- [15] OpenAFS Group, Apr. 2007, retrieved from <http://www.openafs.org>.
- [16] M. Satyanarayanan, "Coda: A highly available file system for a distributed workstation environment," July 15 1999. [Online]. Available: <http://citeseer.ist.psu.edu/239688.html>; <http://www.cs.cmu.edu/afs/cs/project/coda/Web/docdir/wwos2.pdf>
- [17] A. Rai, "Intrinsic security in the SORCER grid," Master's thesis, Texas Tech University, Lubbock, TX, Dec. 2004.
- [18] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," RFC 2459 (Proposed Standard), Jan. 1999, obsoleted by RFC 3280. [Online]. Available: <http://www.ietf.org/rfc/rfc2459.txt>
- [19] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280 (Proposed Standard), Apr. 2002, updated by RFCs 4325, 4630. [Online]. Available: <http://www.ietf.org/rfc/rfc3280.txt>
- [20] C. Ellison and S. Dohrmann, "Public-key support for group collaboration," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 4, pp. 547–565, 2003.
- [21] S. Ajmani, "How to resolve sdsi names without closure," 2002. [Online]. Available: citeseer.ist.psu.edu/ajmani02how.html

- [22] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI Certificate Theory," RFC 2693 (Experimental), Sept. 1999. [Online]. Available: <http://www.ietf.org/rfc/rfc2693.txt>
- [23] *Next Generation Concurrent Engineering*, ISPE. Omnipress, 2005.