Advanced Security Research Journal

VOLUME 1 NUMBER 1 FALL 1998

All inquiries about this publication should be addressed to:
Terry C.V. Benzel, Editor
Vice President for Research, NAI Labs
Network Associates, Inc.
3415 S. Sepulveda Blvd., Suite 700
Los Angeles, CA 90034
310-737-1600
tbenzel@nai.com

Message from the Editor

Welcome to the premier edition of the NAI Labs Advanced Security Research Journal. This edition contains technical reports covering a wide selection of topics in information security. The reports were developed over a several year period, and describe advanced research experiments and prototypes. Future versions of the journal will continue to provide both breadth and depth of coverage through focused reports on a specific topic, such as intrusion detection. In addition, future issues will include guest editor's and solicited reports on emerging technologies of particular interest.

Volume I of the journal includes reports on advanced research performed by researchers at NAI Labs Advanced Security Research division, formerly Trusted Information Systems (TIS). The results of the research projects described are experimental, analytical in nature and often include proof of concept prototypes. All of the research was performed with funding from the Defense Advanced Research Project's Agency (DARPA). This government agency provides funding to the nations top universities and research labs to investigate critical issues facing the DoD and the nation. NAI's Advanced Security Research division has been one of leading recipients of DARPA research contracts in the field of information security. In fact, it was DARPA funded research that lead to the creation of the Gauntlet Internet Firewall.

NAI's Advanced Security Research division continues its long tradition as a research organization, and numerous technologies originally developed under contract to DARPA, are finding their way into NAI's enterprise security products.

Advanced security research topics presented in the journal include:

- Cryptographic topics related to Internet Key Management
- Composable replaceable security services
- Role based access control for firewalls and distributed object systems
- Secure Virtual Enclaves

The first reports, *Internet Key Management and Distribution: Architecture and Toolkit Report*, and *Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization*, describe research in Internet requirements for key management and distribution as they relate to Internet standard protocols as defined in the Internet Engineering Task Force (IETF). The first of these, describes a Key Management and Distribution Toolkit (KMT) that has been designed to facilitate the inclusion of security in applications and protocols. The KMT provides local functions for the generation, storage, protection, and publication of cryptographic keys. The second report presents a new practical algorithm for establishing shared, secret group communications keys in large, dynamic groups. This work addresses the difficult problem of managing cryptographic keys in large dynamically changing groups.

Developing infrastructure components that can be securely combined and replaced in a "plug and play" style will be of increasing importance in the heterogeneous distributed computing environments of the future. The report, *Composable Replaceable Security Services for Survivable Distributed Systems,* describes an infrastructure that applications can use in a manner independent of various operating system and networking technologies. This is accomplished through the development of distributed security services including identification and authentication, confidentiality, integrity, and non-repudiation. These services are provided through two layers of services. The lower level of these services is modeled after the low-level services provided by Intel's Common Data Security Architecture (CDSA) specification.

The report, *Domain and Type Enforcement Firewalls,* reports on research on an enhanced security firewall based on Domain and Type Enforcement (DTE), a strong but flexible form of access control.

A DTE firewall provides several benefits. First it runs application-level proxies in restrictive domains. Second, a DTE firewall coordinates role-based security policies that span networks by passing DTE security attributes between DTE clients and servers. By coordinating DTE security at the endpoints, a DTE firewall adds *defense in depth* to the traditional firewall security perimeter: this permits safe exportation of normally risky services such as NFS. Finally, a DTE firewall interoperates with "non-DTE" systems and associates DTE security attributes with these systems so their interaction with DTE-aware clients or servers can be controlled. This report describe here the design of a prototype DTE firewall system and informally evaluate its security, compatibility, functionality, and performance.

The report, *A Scalable Approach to Access Control in Distributed Object Systems*, extends DTE technology to provide scaleable access control mechanisms for use in distributed object oriented systems such as CORBA or other middleware. The lack of such mechanisms is one of the key obstacles to the widespread use of such systems. Even in large distributed systems it is often desirable to control access to individual objects and methods. In large systems, these are so numerous that the resulting proliferation of access control information can be overwhelming. This report describe Object Oriented Domain and Type Enforcement (OO-DTE), a technology for organizing, specifying, and enforcing access control that has been prototyped and integrated with a commercial ORB and SSL. OO-DTE provides fine-grained control and scalability via a compilable symbolic policy language. The report describes several experimental prototypes and the results of building and using OO-DTE. The OO-DTE technology is also compared with the access control terminology, concepts, and requirements described in Common Object Request Broker Architecture (CORBA) Security Specification.

Collaborative Computing is the sharing of information and services between organizations that collaborate, i.e., have some common mission, goal, or activity. This emerging model of inter-organizational inter-networking is often used to share information with customers, suppliers, or contractors. The report, *Architecture and Design for Secure Virtual Enclaves*, describes early research into defining architectures for secure collaborative computing. Such architectures are based on the concept of a "Secure Virtual Enclave" (SVE). This report describes the requirements for SVE's, the techniques for creating SVE's and discusses architecture and design issues associated with the use of SVE technology in gateways and end systems.

Terry V Benzel

Terry C. V. Benzel
Vice President for Research
NAI Labs

Internet Key Management and Distribution: Architecture and Toolkit Report

by Olafur Gudmundsson, Brian Wellington, David Reeder,
Madelyn Badger, and G. Russ Mundy

1. Introduction

This report describes the architecture and status of the "Key Management and Distribution" toolkit as of December 1997.

The development of the architecture considers a number of areas as they relate to the set of protocols used in the Internet. The Internet Engineering Task Force (IETF) is the principal activity that establishes and publishes standards for Internet protocols. Over the past several years, the IETF has defined security capabilities for several common Internet protocols. In many cases, the defined security capabilities provided essentially no useful management and distribution for cryptographic keys even when the protocols depended on cryptographic functions. When security capabilities included some form of key management and distribution, it was only used by a single protocol. Additionally, available reference implementations do not provide useful human interfaces for essential key management functions, e.g., loading keys into a system. These limitations are impeding the fielding and use of protocols with security capabilities in the Internet. The Key Management and Distribution Toolkit is intended to fill these gaps which should stimulate the effective fielding and use of new Internet security capabilities.

The focus for the development of the Key Management and Distribution Architecture and Toolkit is on Internet requirements for key management and distribution. We are specifying a Key Management and Distribution Architecture that will support a broad range of network security services, promote interoperability and minimize the need for duplication of key management and distribution functions in various protocols. The need for keys and certificates to represent individuals, sessions, hosts, infrastructure and other components as well as the need for multiple representations and authorizations are being addressed. Due to their anticipated wide-spread use and importance for increasing Internet security, we have initially focused our activities on the Domain Name System Security Extensions (DNSSEC) protocol, the Internet Protocol Security Extensions (IPSEC), and the Internet Security Association and Key Management Protocol (ISAKMP). The target audience for this toolkit includes researchers and developers wishing to incorporate key management in their projects.

The KMT (Key Management and Distribution Toolkit) consists of two primary components:

> **libkmt:** a source code library of comprehensive functions to create, maintain, retrieve and use cryptographic keys.
> **user tools:** standalone applications for users to maintain their own key databases, including a graphical user interface.

The guiding principle in the design of the KMT library is to provide an interface that is simple and easy to use. At the same time, libkmt is intended to be a comprehensive library that acts as middleware for a number of crypto packages and key distribution mechanisms as well as providing key publication functions. To allow users to store an arbitrarily large number of keys in their private database, libkmt uses the Berkeley *db* package as the database engine. The user tools, especially the GUI, are designed to provide the ability to use the capabilities of KMT outside of a particular application, and to demonstrate the use of KMT.

The design of KMT is influenced by our experiences in providing keying information for DNSSEC servers and clients. We have also examined a number of crypto packages, both commercial and free, and in most cases key management issues are ignored. We have not found any crypto packages that are able to import or export private keys for use by other packages. KMT addresses this explicitly, as we believe that users may want to share keys between multiple applications. Currently this is only possible if the applications use the same crypto package, and a common storage format.

This report first identifies the approach we have taken in our work, followed by a high level description of the functions provided. The second section covers data structures, data formats, function calls and the overall structure of libkmt. The third section discusses the current set of user tools provided. This is followed by a short description of our current work relating to the integration of KMT with IPSEC components. The report concludes with a short summary and a discussion of future directions.

1.1 Approach

TIS has produced the first version of KMT, designed to facilitate the inclusion of security in applications and protocols. The current version supports the use of KMT by any user level application; work is progressing on creating a kernel module of a subset of KMT that can be used by IPSEC. In addition to that, work is progressing on adding ISAKMP/Oakley capabilities to KMT.

KMT provides local functions for the generation, storage, protection and publication of keys. These functions are applicable to any use of cryptography. For key management specifically in public key cryptography, the toolkit provides for the publication, distribution, and retrieval of keys and validation and authorization structures in DNS. The toolkit will provide for the publication of key certificates as defined in established or proposed public key infrastructures (e.g., PKIX, SPKI, SET, or PGP). The toolkit provides support for the initial configuration of automated symmetric key management systems, as the known systems provide most of the needed publication, distribution, retrieval, etc., functions.

We have developed KMT on common POSIX compliant operating systems and make use of existing freely available software wherever possible. KMT is designed to be portable to any 32 or 64 bit architecture, and is thread safe. KMT has been ported to (but is not limited to) the following platforms:

- FreeBSD 2.2.x
- Linux 2.x
- SunOS 4.3
- Solaris 5.4

KMT supports a range of cryptographic algorithms, parameters and formats in our cryptographic functions for the widest possible applicability. Currently supported algorithms include DES, RSA, Diffie Hellman, MD5, HMAC-MD5, SHA1, HMAC-SHA1, DSA, and IDEA. We have provided links to the following crypto packages to support these algorithms:

- RSAREF
- BSAFE
- libcrypto (from SSLeay)

In a future version, we will include flexibility so that varying policies can be managed. We will work toward international availability of the KMT implementation, with respect to export control restrictions.

KMT is designed to be independent of underlying crypto packages; we provide an indirect and consistent interface to crypto functions through an abstraction layer. This allows us to export KMT without crypto components.

KMT depends on DNS publication as its primary source of public keys. DNSSEC provides security enhancements to DNS, primarily data integrity and authentication using digital signatures on each data set (RR set). Each RR set is signed by the zone key for its domain. A zone's key set is signed by its parent zone at the delegation point, thus building a certification tree.

1.2 Functional Capabilities

1.2.1 Key Generation

KMT provides generation of both symmetric keys and asymmetric key pairs. Any requirements for random numbers in the generation are provided by looking first for a host provided random number device. If that is not present or not enough random data is available, the toolkit will access and combine an assortment of difficult to predict system characteristics in order to provide the necessary level of entropy in the random number generated. Key attributes, such as key size, lifetime and services, are tailorable. The key generation process also checks the key for known attributes of "weak" keys.

1.2.2 Key Storage and Protection

In both symmetric and asymmetric cryptography, there is a portion of the keying material that must be protected from exposure. KMT provides support for storage of the key so that it is available for use but still kept confidential. The fields in the key database are encrypted internally before being written to disk.

Keys retrieved, learned or computed with peers may need to be stored locally. Stored symmetric keys and asymmetric private keys must be protected from exposure. For asymmetric public keys, storage for retrieval and protection from modification is necessary. KMT provides for two levels of storage: a cache for running applications and a permanent database on disk. The disk database is encrypted to protect it from unauthorized access. The cache is not encrypted.

1.2.3 Key Retrieval from Local Storage

Keys are identified by name[1]. There may be several keys associated with one name, varying by algorithm, key size, security service, or application. KMT supports retrieval of a key set (which may be only one key) from the local storage by a footprint, identifier or other attribute, with the choice of the appropriate key from the set left to the application.

1.2.4 Key Expiration and Storage Policies

KMT enforces expiration based on the information gained when the key is generated or retrieved from an external source. Keys are removed from the KMT cache after a short cache lifetime; these keys are not deleted from the database if they have been saved. For example, keys from DNS expire when the TTL of the associated KEY record expires. The KMT database expires keys when the certificate of the key expires. KMT attempts to validate keys that have had their TTL expire but are still in the database; the source identifier is used to check if the key is still valid.

When KMT learns or generates a new key, it is placed in the cache. To add the key to the database, an additional KMT function must be called. Future versions of KMT will allow the storage of expired keys for historical reasons; this is important for keys that are used to sign documents in a specified time period. These keys can be considered to have no expiration but were only considered valid during the specified period.

1.2.5 Key Import and Export

Given the number of different cryptographic algorithms, implementations, and public key infrastructures with specialized key and certificate formats currently proposed for use in the

[1]KMT stores keys with the "name" as the defining characteristic. The name can represent a user, machine, IPSEC SA, etc.

Internet, conversion between key formats will be necessary. KMT will provide for import and export between a set of common key and certificate formats as necessary.

Additionally, KMT can export keys in its own database format for distribution. These files can be encrypted for protection. This method cannot be used to transmit arbitrary data; the KMT database format is defined with a particular format.

1.2.6. Symmetric Key Distribution

When symmetric cryptography is used but automated key agreement is not possible, the symmetric key must be communicated between the peers.

In unicast applications, there are only two peers. KMT will provide support for communicating the symmetric key while protecting it from exposure. We will consider several mechanisms for protected communication of a two party shared symmetric key. Security enhanced electronic mail is one possible mechanism; FTP with security extensions is another. This will support protection of communication between end-users or end-applications as well as protection between end-users and centralized key distribution systems.

In multicast applications, there can be many more than two peers. Multiple unicasts of the symmetric key is possible but inefficient. We will consider efficient mechanisms for the publication of a manually established symmetric group key. A possible method is to communicate a group asymmetric key pair to all participants that will be used to encrypt a group symmetric key. This method would permit new members to join efficiently and for periodic rekeying. The encrypted group symmetric key could be published in DNS. DNS does not serve as an efficient communication medium for pairwise shared secrets, as the communication to and from DNS would offer no saving over direct communication between the pair. In the case of group shared secret keys, particularly where new group members are likely to be added and the group key is likely to change, publication in DNS may offer a savings.

In most cases, automated protection of the symmetric key can only be established by application of asymmetric cryptography or a pre-existing symmetric key. If a key is not available for all peers, other techniques must be used. KMT will provide a number of methods for protected key distribution, which would be considered out of band for other protocols.

The current version of KMT supports both one and two way Diffie Hellman key exchange functions. The Diffie Hellman key exchange process generates a shared secret between the two entities. This shared secret can then be converted into a symmetric key and used for authentication or encryption.

1.2.7 Private Key Distribution

In cases where an asymmetric key pair is generated for a principal but not by the principal, the private key must be communicated to the principal and protected from exposure. The issues for the distribution of private asymmetric keys are the same as for symmetric key distribution. Private key distribution for unicast applications can use the same mechanisms (protected e-mail, FTP with security extensions, etc) as secret key distribution. The distribution of a group asymmetric key can be accomplished by separate communication to each member, protected by individual asymmetric keys or by a group symmetric key.

Automated protected distribution of a private asymmetric key can only be accomplished using pre-existing secure information; a pre-existing symmetric or asymmetric key. When no such key has yet been established, other techniques must be used. KMT will provide a number of methods for protected key distribution. The current version of KMT supports both one and two way Diffie Hellman key exchange functions that can be used to establish a pre-existing symmertic key. KMT may also include facilities for key splitting at later time.

1.2.8 Public Key and Certificate Publication

Publication of the public key involves the following steps:

- Transmission of the public key to the authority who will vouch for the binding between key and identity. This could be a Certificate Authority in some public key infrastructure or a DNS zone, or a trusted key server.

- Creation of binding. For applications that are serving as authorities, the toolkit will provide for the production of a cryptographically protected binding of an identity and a public key. The toolkit will produce this binding in several different public key infrastructure formats, possibly including PKIX, SPKI, SET, or in a DNS KEY record. (Note: any requirement or procedure instituted by the authority for verifying the identity of the key owner will not be a function of the toolkit; it will take place through other means.)

- Publication of the binding. This will entail storage of the binding, whether certificate or KEY record, in a DNS zone.

- Transmission to principal. The binding must be transmitted to the principal associated with the key so that it can be distributed to others in-band with protected messages.

The initial KMT includes functions to facilitate the sequence of actions necessary to publish the key associated with a principal in DNS. First, the key and the principal's associated domain name must be transmitted to the appropriate DNS signing authority (normally the DNS parent zone). The signing authority will generate the KEY set and SIG records and store them in its own zone. The signing authority will also transmit the signed key set back to the principal out-of-band. KMT includes functions to retrieve an X.509v3/PKIX certificate from DNS. Support for other certificate authorities and infrastructures will be addressed in the option task.

For public key management, KMT will provide mechanisms to store and retrieve public key certificates of various infrastructure formats from the DNS. The toolkit will also provide for the use of DNS as an public key infrastructure of its own. The domain name hierarchy provides a ready-made authentication structure for entities that can be named with a fully qualified domain name. Authorization will be provided by proposing a KEYPOLICY record to accompany KEY resource records in the DNS (adherence to the policy will not be the function of the toolkit).

1.2.9 Key Retrieval

KMT provides for the retrieval of a public key from a secure DNS zone, using the provided secure resolver. The toolkit includes support for determining the fully qualified DNS name for certain common identities (e.g., e-mail name for a user on a particular host, Certificate Authorities, host names, user telnet or ftp sessions, etc.) The toolkit also supports retrieval by footprint, identifier or attribute with the choice of the appropriate key left to the application.

Some zones may have a large set of keys to store or frequent requirements to update. Set storage in the zone itself may not be feasible, particularly considering the DNSSEC requirement that the set as a whole must be signed. In that case, there may be a need to provide for a key database that is outside the DNS database. The toolkit will be designed to follow the indirect KEY record in DNS (algorithm 252 - Indirect Key) (Note that some of the DNSSEC security protections would not cover such a key server; any security features relinquished by not using DNSSEC must be provided by the key server).

For those zones that wish to use an off-DNS key server, the indirect KEY record is a mechanism to point at a key server that should be consulted. If an exact match name is not available, the wild-card record would point to the key server.

The initial KMT can retrieve KEY and SIG records directly from the DNS. Support for out-of-zone key databases will be addressed in the option task.

1.2.10 IPSEC

In support of IPSEC, we will support the attribute classes defined in the IPSEC DOI for ISAKMP, which are defined in Section 4.5 of draft-ipsec-ipsec-doi-02.txt.

Using the key, key parameters, key policy and authority information, an IPSEC SA could be automatically formed. Alternatively, retrieving the information from DNS allows the toolkit to construct a security association and insert it into the IPSEC SA database.

1.2.11 Validation

Any public key infrastructure must provide functions to verify the validity of keys. The infrastructure must provide a method to indicate the policy for deciding a key's validity (e.g., validity must be checked on each usage, validity should be checked at the expiration of a stated period, etc.) as well as a method for checking the validity. The KEYPOLICY DNS record will specify the validity policy. The KEYPOLICY record could also contain a pointer to an external validity authority for the key and an access method, much as for the policy engine. It is likely that the policy engine and the authority engine will coexist when both are present. The validity authority will confirm the binding between the key and identity and verify that the key has not been revoked. KMT will provide support for establishing the validity of a key by checking with the validity authority.

In the absence of an external validity authority that can vouch for the validity of the key, the DNS hierarchy itself can be used to determine the validity. In DNSSEC, the DNS signing authority for the zone containing a key record is accessed to determine if the key is among the key set for the zone. This procedure can be used to validate any KEY record in the DNS, not just those used to secure the operation of DNS itself. KMT will provide support for revocation of a key by removal of the associated KEY record.

During the initial deployment of DNSSEC, not all zones will be secure and capable of signing for their delegated zones. A zone that wishes to be secure may find that its parent is not capable of acting as its signing authority. In that case, it will be necessary to have some other zone in the DNS act as the signing authority. The ability to refer in the KEYPOLICY record to signing authorities other than the parent zone will be useful for incremental deployment of DNSSEC.

It is probable that any public key infrastructure will define usage and validity policies for its own certificates. A certificate stored in the DNS will therefore likely be redundant with the KEYPOLICY record. To provide for maximum flexibility, we will allow a KEYPOLICY record to be present even when a certificate is provided. Rules for resolving conflicts between the policy of the public key infrastructure issuing the certificate and the KEYPOLICY record will be formulated.

The initial KMT supports validation of DNS KEY, SIG, and CERT records through a chain of KEY and SIG records from some trusted authority, usually the root name server. We will also support secure denial of existence of X.509 certificates and keys through the DNS SIG and NXT records. Validation of a certificate stored in DNS through some other infrastructure or other trust model will be addressed in the option task. It may not be necessary to introduce the KEYPOLICY resource record for authorization and validation support in the initial KMT.

2. KMT Library

Figure 1 shows the relationship between different components of the KMT library. It is described below, in order of increasing abstraction.

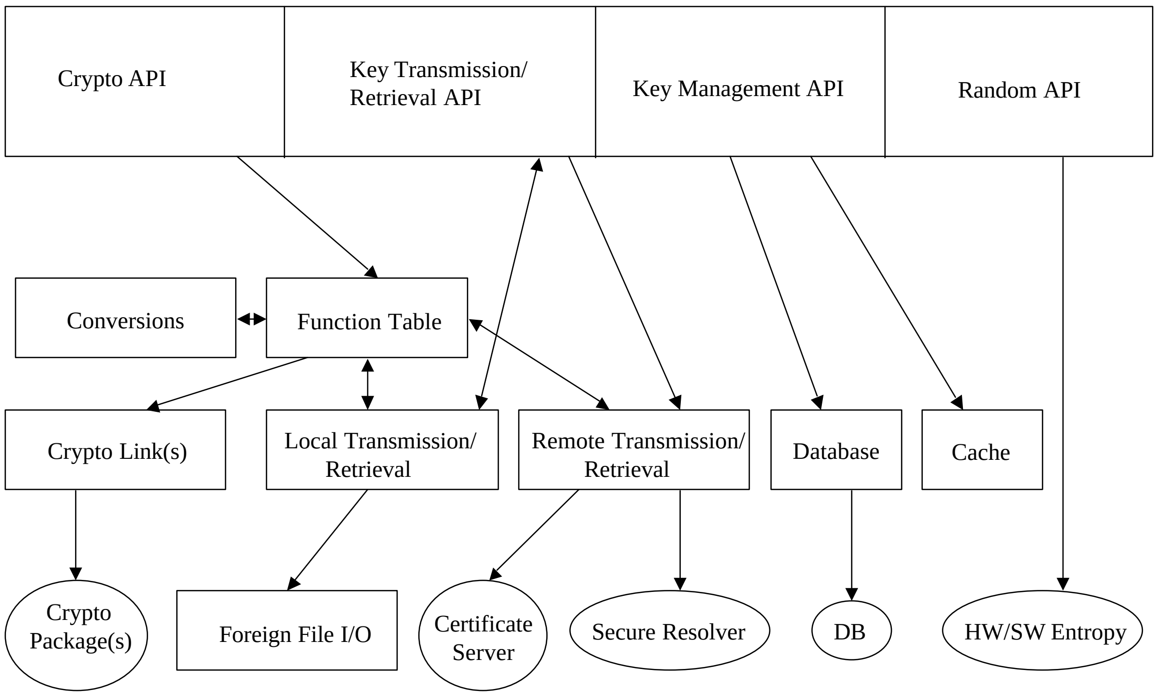

Figure 1: KMT Library Components

- Underlying subsystems

Crypto Package(s) Different packages are used to provide implementations of cryptographic algorithms.
Foreign File I/O KMT can read and interpret keys produced by other applications.
Certificate server KMT can communicate with certificate servers of different infrastructures.
Secure resolver The TIS secure resolver, developed as part of DNSSEC and KMT, is used to retrieve information from DNS and check its validity.
DB The Berkeley *db* 2.0 package is used to provide a fast, portable database, and included with KMT.
HW/SW Entropy If available, /dev/random is used to provide hardware entropy. If not, a series of pseudo-random operations querying system state are performed.

- Internal subsystems

Crypto links For each algorithm provided by each crypto package, there is an abstraction layer defined, so that all packages can be used identically. The abstraction layer also provides stub functions for unimplemented functions; whether they are undefined operations or unsupported by the implementation.
Local transmission/retrieval KMT provides a standard interface for use of foreign file formats.
Remote transmission/retrieval KMT provides a standard interface for communication with other distribution mechanisms, using available protocols.
Database KMT extends *db* to provide a simple interface to a database, encrypted for protection. Sets of keys, distinguished by name, are stored as a single record.
Cache The KMT cache is based on a hash table, and is used to provide fast access to recently used and/or obtained keys.

Conversions For each algorithm, there are routines defined to convert between the KMT portable format and other formats.

Function table The KMT function table stores pointers to all conversion and cryptographic routines for all algorithms. For each algorithm, the cryptographic routines are chosen from the "best" available package for the algorithm.

- API

Crypto API KMT provides functions to use keys to encrypt/decrypt and authenticate/verify data. KMT can also be used to generate keys and create keys based on existing data, as well as performing Diffie Hellman key exchange calculations.

Key Transmission/Retrieval API KMT provides functions to import and export keys and/or certificates.

Key Management API KMT provides functions for operations on keys in the cache and/or database. Keys can be viewed, deleted, saved, compared, and selected based on attributes.

Random API KMT provides a function to allow applications to request random data, using its source of entropy.

2.1 Data Types

KMT exports two data structures to applications: an opaque key structure and a key set (key list) structure. **KMT** functions operate on both of these structures and, in many cases, the internal data structures dependent on the crypto package used.

A **KMT** database entry stores, for each key, a number of parameters related to the type of the key as well as encoding of the key in an algorithm specific portable format that encodes each field as a type/length/data tuple. The **KMT** storage functions also perform byte order conversions: keys in the database are stored in network byte order, while keys either in the cache or stored by the application (in **KMT_KEYs**) are stored in host byte order.

2.1.1 KMT_KEY

Keying material and related information. This structure is opaque to the user (the type definition is not in the exported header file), but the following fields can be obtained:

Name The name that the key is stored under.

Key data A pointer to crypto-package specific keying material.

DNS Flags Defined by DNSSEC (RFC 2065 and successors).

DNS Extended Flags Defined by DNSSEC. Currently unused.

Expiry Whether a key can expire at a certain time or after a certain amount of usage.

Expiration time Circular time value storing the expiration time of a key. If no_expire is set, the key does not expire.

Expiration usage The number of bytes that a key can operate on before expiring.

Size The size of the key, in bits.

DNS Footprint The key's footprint. Used to quickly determine whether keys may be identical, as defined by DNSSEC.

Algorithm The key's algorithm.

Algorithm mode Addition information about the key's usage, such as padding, feedback mode, etc.

DNS Protocol The protocol, as defined by DNSSEC.

Type Public Key, Private Key, Symmetric Key, Certificate, or Diffie Hellman parameters.

File source Origin of the key. Each foreign file format has a unique source entry; there are also unique values for generated keys and specified keys, as well whether the key originated during this session or was extracted from the database.

2.1.2 KMT_KEY_LIST

A doubly linked list representing a set of keys.

kmt-key pointer to a **KMT_KEY** record

Prev,Next pointers to the previous and next **KMT_KEY_LIST** record.

2.1.3 KMT Database Format

A record in the **KMT** database corresponds to a set of keys with the same name; it is constructed from a **KMT_KEY_LIST**. Each key in the set is converted to an algorithm specific portable format, and then the portable keys are concatenated into a single block of data.

A KMT_KEY has two main parts: the non-cryptographic data and the pointer to crypto package specific data. The non-cryptographic data is common to all algorithms, and is simply converted to network byte order and padded to a 64 bit boundary for storage.

The crypto package specific data is more complicated. For each algorithm, a common set of parameters is defined; all packages must either internally store or be able to generate all of these fields, and all packages must be able to recreate a key after reading these values. Each saved parameter type is assigned a unique 16 bit identifier. Each parameter is converted to a bit string. For each parameter, the database stores the unique identifier, the length of the bit string (in bytes), and the bit string (padded to a byte boundary). These vectors are then appended to the converted non-cryptographic data defined above. For each algorithm, there is a defined order for the parameter vectors; **KMT** can understand records where these are out of order, but for internal use, the order is important (conversions from foreign file format to **KMT** portable format to **KMT** internal format rely on **KMT** generating portable format with vectors in the correct order).

2.2 Storage

2.2.1 Application

When keys are put into the **KMT** cache, the API function returns a list of references to the keys. These references are then used as parameters to **KMT** API routines. Multiple references can exist pointing to the same key in the cache; each key stores a reference counter and is not deleted until both a delete has been requested and there are no outstanding references.

2.2.2 Cache

The **KMT** cache is based on a hash table. Each bucket contains a pointer to a list of cache nodes, which each contain a list of all keys for a name. When an application using **KMT** exits, all keys in the cache marked as being associated with the database are written; all others are lost.

2.2.3 Database

The KMT database uses the Berkeley *db* library. A database record is constructed from a list of keys, and is stored in an architecture independent format. The entire block of data (as constructed above) may be encrypted with a DES key for increased security - the same key is used to encrypt all records in a database file, and is not stored on disk. For the default **KMT** database, the encryption key is constructed from a hash of the password supplied to **kmt_init()**.

2.3 Constants

2.3.1 Cryptographic Operation Modes

KMT_CRYPT_MODE_INIT
KMT_CRYPT_MODE_UPDATE
KMT_CRYPT_MODE_FINAL
KMT_CRYPT_MODE_ALL

Cryptographic operations may be performed in multiple steps. An **INIT** must be performed, followed by one or more **UPDATE**s, followed by one **FINAL**. These flags can be combined, so that **UPDATE** may be done along with **INIT** or **FINAL**. As a shortcut, **KMT_CRYPT_MODE_ALL** is a synonym for the union of the other 3 operations.

2.4 API

The Key Management Toolkit attempts to provide a consistent API supporting many common operations. All routines return **KMT_ERR_SUCCESS** if the operation completed successfully, and an error code (See Return Codes, Section xx) on failure, unless otherwise noted.

In several of these functions, a block of data is returned. A set of 2 parameters is passed into the function: a (u_int8_t **) pointing to the data and an (int *) pointing to the length. If the data is non-NULL and the length is not 0, the storage is assumed to be allocated by the application, and have the supplied length. Otherwise, KMT will allocate space itself, and use the parameters to return the location of the data and its length. If the length is supplied, but insufficient, the **KMT** routine will return **KMT_ERR_INSUF_STORAGE**.

2.4.1 Setup

2.4.2

```
kmt_init(const char *pass, const char *db, int
      (*func)(const char *, ...))
```

kmt_init() initializes KMT. The cache structure is allocated. The key database db is opened, and a DES key is constructed from a hash of pass, which is used to encrypt the database for protection; if the database file exists, the password is validated by successfully decrypting a record from the file. The final parameter is a pointer to a function used for error reporting; this defaults to printf(3) if not specified. All KMT routines will return **KMT_INACTIVE** if **kmt_init()** has not been successfully called.

2.4.2 Shutdown

```
void kmt_close()
```

kmt_close() frees all memory associated with KMT (the cache) and closes the database. This function is registered with atexit(3), so it will run automatically on program exit, but can also be called explicitly to free KMT's memory earlier than program exit.

2.4.3 Key Creation

Routines for creating new keys.

```
kmt_generate_key (char *name, u_int16_t keylen,
        u_int16_t alg, int alg_mode,void *parms,
        KMT_KEY_LIST **list)

kmt_specify_key (char *name, u_int16_t keylen,
        u_int16_t alg, int alg_mode, void *parms, u_int8_t *data, int
        len, KMT_KEY_LIST **list)
```

These routines create keys and enter them into the KMT cache. The key name, length, algorithm, and algorithm mode are supplied. In addition, any algorithm specific data needed for key creation is passed through the parms parameter. **kmt_generate_key()** uses random data to generate a key with the given characteristics, while **kmt_specify_key()** uses the supplied data and length to construct a key. **kmt_specify_key()** is only currently defined for some algorithms, and will probably only be defined in the future for algorithms containing exactly one data field in the key. Both of these routines return a list of references to all keys entered into the cache in a parameter.

2.4.4 Key Information

```
kmt_key_name(KMT_KEY *key, char **name)
```

kmt_key_name() returns the name of the key in a parameter. The pointer points directly at the name in the key structure; no memory should be pre-allocated and no memory is allocated in this function.

```
kmt_key_attr(KMT_KEY *key, int attr, int *value)
```

kmt_key_attr() returns the specified attribute of the key in a parameter. No memory should be pre-allocated and no memory is allocated in this function.

```
kmt_key_attrs(KMT_KEY *key, KMT_ATTRIBUTE *attr,
         int nattr)
```

kmt_key_attrs() returns the specified attributes of the key. The attr array is configured with the desired attributes in the type fields, and fills in the value fields.

```
kmt_set_dnsinfo(KMT_KEY *key,
      const u_int16_t dns_flags,
      const u_int16_t ext_flags,
      const u_int8_t dns_proto)
```

This routine sets the DNS flags, extended flags, and protocol fields of a key.

```
kmt_set_expiry(KMT_KEY *key, const u_int32_t
      expire_time, const u_int64_t expire_usage)
```

This routine sets the expiration policy for the key. If a nonzero time is specified, the key is set to expire at that time. If a nonzero usage is specified, the key will expire after operating on that number of bytes.

2.4.5 Cache Interaction

Routines for access to keys in the KMT cache.

```
kmt_get_keylist_from_cache (KMT_KEY_LIST **listp,
        char *name, kmt_attribute *attr, int nattr)
```

```
kmt_va_get_keylist_from_cache (KMT_KEY_LIST **listp,
        char *name, ...)
```

These routines retrieve information from the KMT cache, in the form of a list. The key's name is specified, as well as any additional attributes to be selected on, and references to all matching keys are returned. If keys have expired, they will be marked for deletion. If keys are marked for deletion and have no outstanding references, they will be deleted. No keys marked for deletion will ever be returned to the user.

kmt_get_keylist_from_cache() is passed an array of attributes (type/value pairs) and the size of the array; **kmt_va_get_keylist_from_cache()** takes type/value pairs as parameters, and signifies the end of the list with a NULL parameter (See section 2.6).

```
kmt_list_cache (KMT_KEY_LIST **list)
```

This routine returns a list of all active keys in the KMT cache. It is designed to be used for displaying a list of all keys, but is not restricted to that.

```
kmt_clear_cache()
```

Removes all keys from the KMT cache. This should be used with caution, as references to keys can be made invalid by this call.

```
kmt_age_cache()
```

Marks all expired keys in the **KMT** cache for deletion. If a key is marked for deletion and has no outsanding references, its memory is freed.

2.4.6 Database Interaction

Routines for access to keys in the **KMT** database. Some of these routines will also read/update the cache.

```
kmt_get_keylist (KMT_KEY_LIST **listp, char *name,
        kmt_attribute *attr, int nattr)
```

```
kmt_va_get_keylist (KMT_KEY_LIST **listp,
        char *name, ...)
```

These routines retrieve information from the **KMT** cache and database, in the form of a list. The key's name is specified, as well as any additional attributes to be selected on. All keys with the selected name are read from the database into the cache at the start. References to all matching keys in the cache are returned. If keys have expired, they will be marked for deletion. If keys are marked for deletion and have no outstanding references, they will be deleted. No keys marked for deletion will ever be returned to the user. **kmt_get_keylist()** is passed an array of attributes (type/value pairs) and the size of the array; **kmt_va_get_keylist()** takes type/value pairs as parameters, and signifies the end of the list with a NULL parameter. (See).

```
kmt_delete_key (KMT_KEY *key)
```

Deletes a key from the **KMT** cache. If the key is associated with the database, it is removed from there also. As the key will likely be part of a keylist, this routine does not free any data structures in the cache, it marks the key for future deletion when there are no outstanding references.

```
kmt_write_key(KMT_KEY *key)
```

This routine writes a cached key to the **KMT** database. This key is also marked as associated with the database, so it will be updated on kmt_close.

```
kmt_load_database()
```

Loads all keys from the **KMT** database into the cache. This can be called at startup, or when a traversal of all permanent keys is desired.

```
kmt_clear_db()
```

Removes all keys from the **KMT** databse. Any keys present in the database and not in either the cache or the application's memory will be lost.

2.4.7 Key I/O

Routines for converting between **KMT** and other formats.

```
kmt_input_key (const char *file, int type,
        KMT_KEY_LIST **list)
```

Reads a key from a foreign file format. The name and type (See) of the file are supplied. The keys are converted to **KMT** format and stored in the cache, and references are returned to the user.

```
kmt_output_keylist (const KMT_KEY_LIST *list,
        const char *file, int type)
```

Writes a set of keys to a foreign file format. The key list is supplied, along with the name and type (See 2.7) of the file.

2.4.8 DNS Support

Routines for interacting with a DNS server, building a message for publication, and converting between DNS and **KMT** formats.

```
kmt_get_keylist_from_dns (char *name,
        KMT_KEY_LIST **list)
```

Retrieves all keys associated with the supplied name from DNS. A secure resolver is used to guarantee the validity of the data. The keys are converted into **KMT** format and stored in the cache, and a list of references is returned.

```
kmt_build_dns_publication_message (char *name,
        KMT_KEY *key, u_int32_t expiration, char **text)
```

Constructs a signed set of keys to be stored in the DNS. All public keys with the specified name are added to this set, and the supplied key is used to cryptographically sign it. The expiration refers to the signature's expiration. The signed set is returned as a text string, which the application can handle (i.e. mail to the DNS administrator, add to a zone file, etc).

```
kmt_to_dns (KMT_KEY *kmt_key, int *out_len,
        u_int8_t **dns_key)
```

Converts a key to an in-memory DNS KEY record.

```
    kmt_from_dns (const char *name, const int in_len,
            const u_int8_t *dns_key, KMT_KEY_LIST **kmt_key)
```

Converts an in-memory DNS KEY record into a **KMT** key. The key is stored in the cache, and a list (of length 1) of references is returned.

2.4.9. Cryptographic Operations

Routines to perform cryptographic operations using keys. The keys are not required to be stored in either the KMT cache or database.

These operations are all passed a mode parameter, which allows the operation to be performed in its entirety or in parts (init/update/final). If multiple calls are used, context is used to maintain state; it will be initialized if (mode==INIT) and freed if (mode==FINAL). If not, context is unused and can be NULL.

```
    kmt_sign_data (const int mode, void **context,
            const u_int8_t *data, const int data_len,
            KMT_KEY *private_key,u_int8_t **signature,
            int *sign_len)
```

Using the supplied key, compute a digital signature of the supplied data.

```
    kmt_verify_data (const int mode, void **context,
            const u_int8_t *data, const int data_len,
            KMT_KEY *public_key, u_int8_t *signature,
            int sign_len)
```

Using the supplied key, verify the digital signature of the supplied data.

```
    kmt_encrypt_data (const int mode, void **context,
            const u_int8_t *data, const int data_len,
            u_int8_t **enc_data, int *enc_data_len,
            KMT_KEY *public_key, u_int8_t **parms)
```

Using the supplied key, encrypt the supplied data. Any algorithm specific data needed for encryption is passed through the parms parameter, and any necessary algorithm specific results (such as a randomly generated initialization vector) are returned in parms.

```
    kmt_decrypt_data (const int mode, void **context,
            const u_int8_t *data, const int data_len,
            u_int8_t **dec_data, int *dec_data_len,
            KMT_KEY *private_key, u_int8_t **parms)
```

Using the supplied key, decrypt the supplied data. Any algorithm specific data needed for decryption is passed through the parms parameter.

2.4.10. Diffie Hellman Key Exchange Specific Routines

KMT can be used to perform a Diffie Hellman key exchange. There are several ways to get started: KMT will generate a DH parameter set from scratch, or use DH parameters from an input message, an existing DH key structure, or a DNS DH KEY record. If the parameters are generated from scratch, a message is also generated that can be used to initialize potential DH partners with the same parameters.

Once a key structure is initialized with DH parameters, an Exchange can be started. In this step a random private value and a public value are generated and saved. An Exchange Message containing the public value is generated and returned.

Two applications are initialized to the same values, both start DH exchanges, each is given the Exchange Message generated by the other, and in the final step, they derive a shared secret from their stored private value and the public value in the Exchange Message.

A one-way DH exchange can be accomplished if an application initializes, starts a DH exchange and publishes information in the DNS that can be used to derive the initialization parameters and the Exchange Message. Any other application can use the DNS DH KEY record to initialize, start and complete an exchange. The Exchange Message produced is sent to the DNS key owner who can derive the same shared secret.

```
    kmt_dh_generate (char *name, int alg_mode, void *parms,
                 u_int8_t **msg, u_int16_t *msg_len,
                 KMT_KEY_LIST **listp)
```

This routine generates a set of Diffie Hellman parameters with the properties specified in parms, initializes a key structure with these parameters, and returns a message that can be used to initialize a DH partner with the same parameters. The input parameters can be filled in a DH_PARAM_GEN_PARMS structure and then passed in pointed to by parms. There are three values: Primebits, Expbits and Generator. The Primebits can be anything (256 makes a good test value). A larger prime is more secure, but takes longer to process. Expbits, the size of the private exponent, can be any value < Primebits. Expbits may be required input (bsafe) or default (Eric Young uses Primebits-1). The Generator, or Base, may be created by the crypto package (bsafe) or required input (Eric Young's SSL crypto). It can be any integer value, but 2 and 5 are recommended defaults. To be crypto package independant it is best to set all three values.

```
    kmt_dh_msg_init (int alg_mode, u_int8_t *msg,
             u_int16_t msg_len, KMT_KEY_LIST **listp)
```

Initialize a key structure with DH parameters from an input message. This can be used to initialize a DH exchange with the same parameters as those being used by a DH partner.

```
    kmt_dh_key_init (char *name, int alg_mode,
             KMT_KEY *use_key, KMT_KEY_LIST **listp)
```

Initialize a key structure with DH parameters stored in another Diffie Hellman key structure. This can be used when multiple key exchanges will be performed with the same initial parameters.

```
      kmt_dh_exch_from_dns (char *name, u_int8_t *dns_key,
            const int in_len, u_int8_t **exch_msg,
      u_int16_t *exch_msg_len, u_int8_t **secret, u_int32_t
            *secret_len, KMT_KEY_LIST **listp)
```

This routine is supplied with a name and a DNS DH KEY record It performs a one-way Diffie Hellman exchange, and returns a key structure, a DH Exchange Message and a shared secret. If the owner of the DNS DH KEY record is given this DH Exchange Message, it can derive the same shared secret.

```
    kmt_dh_start_exchange (KMT_KEY *dh_key, u_int8_t **msg,
                 u_int16_t *msg_len)
```

This routine takes a key structure initialized with DH parameters and generates a random private value and a public value. These values are stored in the key structure. A DH Exchange Message containing the public value is generated and returned.

```
kmt_dh_complete_exchange (KMT_KEY *dh_key,
              const u_int8_t *msg,
              const u_int16_t msg_len,
              u_int8_t **secret, u_int32_t *secret_len)
```

This routine takes an Exchange Message from a DH partner and a key structure that has been initialized with DH parameters and has generated its own public and private values. The DH exchange is completed and a shared secret is returned.

2.4.11. Translations

These routines take a KMT defined numeric constant, and return a short textual description.

```
char *kmt_error_str (int error)
char *kmt_alg_str (int alg)
char *kmt_type_str (int type)
char *kmt_source_str (int source)
char *kmt_dns_flags_str (int dns_flags)
char *kmt_proto_str (int dns_proto)
char *kmt_alg_mode_str (int mode)
```

2.4.12. Other Support Utilities

```
  kmt_list_supported_algorithms(u_int16_t **algs,
            int *nalgs)
```

In the user specified array, denote all algorithm identifiers supported by KMT, and the size of this list.

```
kmt_compare_keys (const KMT_KEY *key1,
              const KMT_KEY *key2)
```

Compares two keys. This determines if the two keys have the same name, flags/ext flags, protocol, algorithm, size, footprint, type, and internal representation of cryptographic data, and returns KMT_SUCCESS if the keys are identical and KMT_FAILURE otherwise.

```
kmt_compare_keylists (const KMT_KEY_LIST *list1,
              const KMT_KEY_LIST *list2)
```

Compares two keylists. This determines if the two lists contain the same keys (as defined by **kmt_compare_keys()**. This returns KMT_SUCCESS if the lists are identical and KMT_FAILURE otherwise.

```
kmt_copy_key (KMT_KEY **key1, const KMT_KEY *key2)
```

Creates another reference to the key referred to by key2 and assigns it to key1.

```
kmt_release_key (KMT_KEY *key)
```

Releases a reference to a key. This will not free the key unless the key has been marked for deletion and has no other outstanding references.

`kmt_release_keylist (KMT_KEY_LIST *list)`

Releases a list of references to keys. This will not free any keys unless a key has been marked for deletion and has no other outstanding references.

`kmt_random (u_int8_t *data, int size)`

Generates random data. This uses /dev/random (hardware generated entropy) if possible, and otherwise performs a series of pseudo-random operations querying system state.

2.5 Return Codes

KMT-ERR-SUCCESS The operation was successful.
KMT-ERR-FAILURE The operation failed.
KMT-ERR-LOCKED The KMT thread lock was set.
KMT-ERR-NULL-FUNCTION An unimplemented function was called, or the function table is corrupted.
KMT-ERR-INACTIVE KMT has not been properly initialized.
KMT-ERR-NODATA The requested name has no keys.
KMT-ERR-NOKEY There is no key matching the given specifications.
KMT-ERR-DNS-NULL-KEY The key read from DNS is a null key (cannot sign or encrypt).
KMT-ERR-NULL-KEY The specified key is NULL.
KMT-ERR-NULL-LIST The specified key list is NULL.
KMT-ERR-NULL-DATA The specified data is NULL.
KMT-ERR-INVALID-ALG The specified key has an unsupported/invalid algorithm.
KMT-ERR-INVALID-OP This operation cannot be performed by the specified key, due to its flags.
KMT-ERR-INVALID-INPUT An input parameter is invalid. This should be replaced with a more descriptive error.
KMT-ERR-INSUF-STORAGE Insufficient storage was supplied. Either a larger buffer should be provided, or the routine should be told to allocate space itself.
KMT-ERR-NULL-CONTEXT The context state variable is NULL when it should have a value (during a non-INIT cryptographic operation).
KMT-ERR-ALLOC An allocation or initialization failed. This is always caused by an underlying subsystem failure.
KMT-ERR-INVALID-PARMS The algorithm specific data is invalid.
KMT-ERR-INVALID-KEY The key type is invalid for the algorithm or operation.
KMT-ERR-GENERATE-FAILURE Key generation failed.
KMT-ERR-IPSEC-KEY-TOO-SHORT The specified IPSEC key is not long enough.
KMT-ERR-TRUNC An object contains less data than it should.
KMT-ERR-UNSUPPORTED-MODE The specified algorithm mode is unsupported.
KMT-ERR-MISALIGNED-INPUT The data is not aligned to a valid number of bytes.
KMT-ERR-SPECIFY-RAND-ONCE A key cannot be specified and have a random initialization vector.
KMT-ERR-DECRYPT-RAND Data cannot be decrypted with a random initialization vector.
KMT-ERR-UNDEFINED-ATTR A selected attribute is undefined.
KMT-ERR-EXPIRED-KEY The selected key has expired.
KMT-ERR-INIT-FAILURE The INIT mode of a cryptographic operation failed.
KMT-ERR-UPDATE-FAILURE The UPDATE mode of a cryptographic operation failed.
KMT-ERR-FINAL-FAILURE The FINAL mode of a cryptographic operation failed.
KMT-ERR-INVALID-DH-STATE The Diffie Hellman exchange state is invalid.
KMT-ERR-INVALID-VECTOR An encoded field of a key in the database is invalid.
KMT-ERR-INVALID-RECORD A record in the database (keylist) is invalid.
KMT-ERR-DB-AUTH The record in the database fails authentication .
KMT-ERR-DB-ERROR The database library returned an error.
KMT-ERR-CRYPTO-ERROR The crypto package returned an error.

2. 6. Attributes

See 2.1.1. for more information.

KMT-ATTR-FOOTPRINT Footprint
KMT-ATTR-PROTO Protocol
KMT-ATTR-ALG Algorithm
KMT-ATTR-ALG-MODE Algorithm mode
KMT-ATTR-SIZE Size (in bits)
KMT-ATTR-MINSIZE Minimum size
KMT-ATTR-MAXSIZE Maximum size
KMT-ATTR-TYPE Key type
KMT-ATTR-FLAGS Flags; matches keys with exactly these flags set.
KMT-ATTR-FLAGS-SET Subset of flags; describes keys with at least these flags set.
KMT-ATTR-EXT-FLAGS Extended flags
KMT-ATTR-SOURCE Source
KMT-ATTR-EXPIRE-TIME Time when the key expires
KMT-ATTR-EXPIRE-USAGE Number of bytes remaining to operate on before expiring.
KMT-ATTR-EXPIRY Expiration method

2.7 Foreign File Formats

The following formats are supported by KMT:

PGP: PGP keys can be stored in KMT. However, they are signed with the IDEA algorithm, which is unsupported. Thus, PGP keys cannot be verified on load, and cannot be written (under development).
SSH: Host SSH keys can be stored in KMT, and written to disk. Users' SSH keys will also be able to be read and written (under development).
PFKEY: KMT will be able to communicate using PFKEY.
X.509/PKIX: Certificates will be read and interpreted by KMT (under development).
SPKI: Certificates will be read and interpreted by KMT (under development).
KMT: KMT can create additional databases which can be transmitted to other users and/or instances of KMT.
DST: KMT can read and store keys created by TIS's Digital Signature Toolkit. If needed, support can be added to write these files.
DNS: DNS KEY records can be read and stored by KMT.

2.8 Function Table/Crypto Interface

KMT uses a multi-level interface to cryptography, centered around a global table of per-algorithm functions. At compile time, all available crypto packages are registered. At run time, KMT determines (using a set of predefined rules) which algorithms should be implemented using each package for maximum efficiency.

The core cryptographic functions are implemented on a per-package basis. Thus, each crypto package will have a suite of functions for each supported algorithm, supporting cryptographic operations such as generate/specify, sign/verify, encrypt/decrypt. Also, there are functions for standard operations such as copy, compare, free, and conversion to the KMT architecture and package independent (but algorithm specific) portable format. KMT internally represents keys in the package's native format, so most of these functions are fairly simple.

Higher level functions are implemented on a per-algorithm basis, such as conversion to foreign file formats. Instead of converting these keys directly into KMT's internal representation, they are first converted to portable format, and then from portable format into internal form.

Each key stores a pointer to a row of the function table, which contains functions appropriate for a key of that algorithm. This table includes both the per-package and per-algorithm functions, and is initialized by kmt_init(). When the key is created, its pointer into the function table is assigned, and is persistent for the key's lifetime.

3. KMT User Tools

Figure 2 on the next page shows the relationship between different components of the KMT User Tools. It is described below.

KMT Library Described in Section .
KMT Daemon Used to provide text based access to the library.
(KMT Daemon) Multiple KMT Daemons can communicate with each other.
CGI Scripts These provide the back end for the GUI interface.
Command Line Tools Standalone programs performing some of the functionality of KMT are provided.
GUI A web browser displays pages where users can perform KMT operations.

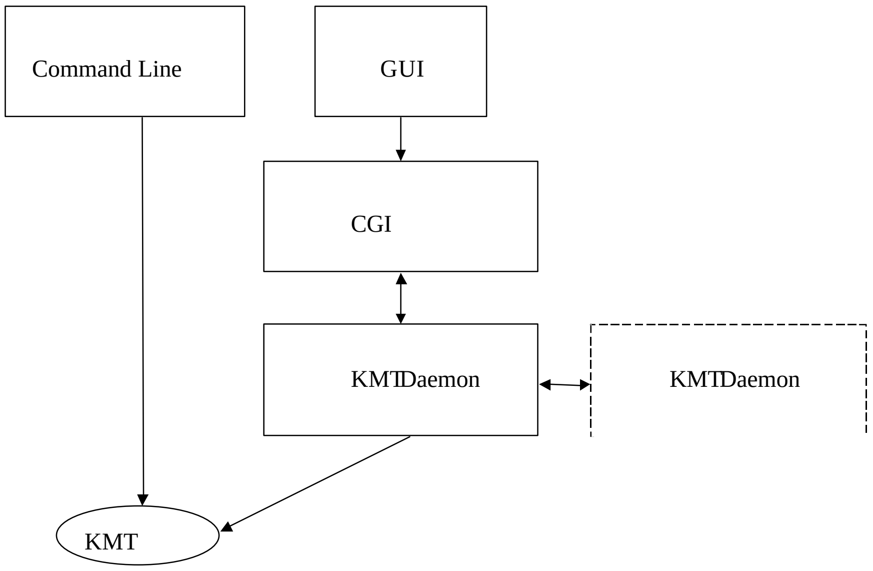

Figure 2: KMT User Tools

3.1 *KMT Daemon*

3.1.1 Design

The KMT Daemon is a simple wrapper around the KMT API functions. It reads text-based commands corresponding to KMT library calls over a UNIX domain socket, and executes the routines. All output is sent as text through the same socket, where it can then be processed by the sending process. The daemon is designed for use by scripts and/or communicating with non-KMT aware programs.

3.1.2 Capabilities

The daemon provides access to all KMT library routines, as well as additional functionality. Real-time Diffie Hellman key exchanges are supported. Two independent KMT daemons can communicate and arrive at a shared secret. One daemon generates the initial keying material. The second daemon, after being supplied with the location of the first daemon, makes a TCP connection to it (on a defined port). A parameter initialization message can be sent if the DH parameters are not well known, then a DH exchange message is sent by each daemon to the other. The KMT library DH routines compute a shared secret based on these messages and the local private data.

3.2 KMT GUI

The KMT GUI is implemented using a web browser interface. There are a number of HTML forms allowing the user to enter data, and the data is processed by perl5 CGI scripts. The scripts pass messages to and from the KMT daemon.

The interface is designed to be easy to use and consistent, and support all of the functionality of the KMT library. All operations can be accessed from the main page, and most can also be accessed from subpages. When an operation is performed, it will indicate whether it was successful or not. If unsuccessful, there will be enough information so that the user can correct the mistakes and retry the action.

3.2.1 Login

When loading the top level web page, the user enters a password and a database file name (the password is hashed into a DES key used to encrypt the database). Assuming the password is correct, the user now has access to the database file. At login time, there is the option of reading the entire database into the cache.

If a KMT daemon is currently running, the login procedure will connect to it. If not, a daemon is started, and continues to run unless it is explicitly shut down.

3.2.2 Start/Shutdown

The KMT daemon can be manually started from this page (the user still must log in to use any of the functionality) or can be shut down.

3.2.3 Generate keys

The user specifies a key name, algorithm, size, and any other appropriate parameters, and a key will be generated and stored in the KMT cache. The key protocol and flags can also be set, as well as its expiration time.

3.2.4 Sign/Encrypt

A key, specified by name, can be used to sign and/or encrypt a block of user entered text; the resulting data is encoded in hex and displayed in the browser.

3.2.5 Diffie Hellman Exchange

Initiate Diffie Hellman Exchange: The user of the initiating server enters the key name, size, and optionally flags, protocol, and expiration time. After Submit is selected, the daemon waits for a connection.

Associate Diffie Hellman Exchange: The user of the associating server enters the initiating host name, key name, size, and optionally flags, protocol, and expiration time. This server contacts the initiating daemon to establish a connection, and the protocol for Diffie Hellman key exchange is performed.

After the exchange, each server has a copy of the shared secret. The user can then save the secret as a key in the KMT cache, using any algorithm supported by **kmt_specify_key()**.

3.2.6 List Supported Algorithms

The server displays a list of all supported algorithms.

3.2.7 Build DNS Publication Message

The user specifies a key set to be prepared for inclusion in a signed DNS zone; this process extracts all public keys with that name. There is also a key specified to sign the set of keys; this may or may not have the same name. The result is displayed in the browser, where it can be saved and entered directly in a DNS zone file or sent to the local DNS administrator.

3.2.8 Write Key to Database

The user specifies a key set to be encrypted and stored in the database.

3.2.9 View Cache

The user enters a key name; all keys with this name are displayed on screen in a descriptive format. Alternatively, if the name "all" is specified, all keys in the cache are displayed.

3.2.10 Load Database

All keys stored in the database are loaded into the cache. This can also be done by checking the "Load" button when performing a login.

3.2.11 Import Foreign Key

The user selects a file on disk containing keys, and the format of the file. The file is read, and the keys are converted to KMT's internal format and stored in the cache under appropriate names (either obtained from the keys themselves or the filename).

3.2.12 Export Key

The user selects a key set, and an output file name and format. KMT converts the keys to the appropriate format and creates the file.

3.2.13 Incomplete Capabilities

Delete Key A set of keys is displayed, and can be marked for deletion.

Retrieve Key from DNS A secure query for a given name is sent to the DNS server; these keys are converted into KMT's internal format and stored in the cache.

Transmit a Secret Key to a Group A key set and recipient are selected, and a secure transport method is used to send the key.

4. KMT Support of IPSEC SAs and PF_KEY

Recent development work has focused on adding support for IPSEC in KMT. Changes are necessary to support the PF_KEY messaging protocol and to add support for IPSEC Security Associations (SAs). In addition, we are determining whether to create a version of KMT that would reside inside the IPSEC kernel, or which would allow the kernel to communicate with a KMT key server over the PF_KEY messaging "bus". We will build a prototype using the NRL IPSEC kernel and associated tools.

4.1 IPSEC SA Representation

From the design perspective of KMT, IPSEC SAs constitute keys with state. The KMT_KEY structure (documented in section) must be slightly modified to support IPSEC “keys”. IPSEC SAs associate new characteristics with KMT key material including:

- states *in between* active/inactive such as **unique** and **larval**
- key use lifetime which may be measured by bytes in addition to time
- sub-types for attribute classes such as AH, ESP, MOBILEIP, etc.

These characteristics will be incorporated into the current definition of a KMT_KEY.

IPSEC keys will also need a consistent naming scheme that reflects the following information:

- end-point and interface addresses
- the security Parameter Index (SPI)

4.2 Policy Management for IPSEC

The current version of KMT does not implement a policy mechanism although the IPSEC IETF documents mandates support for one. As we consider the ramifications of using the KMT Toolkit to simplify the needs of the NRL IPSEC kernel, we will also consider adding a mechanism which will allow the expression of the mandated range of security policies.

5. Conclusions and Further Work

The first version of KMT will be released in February 1998. This version will contain enough functionality to support many applications.

In the next year, we will add policy control to KMT, as well as support for group keys and additional protocols.

Work funded by DABT63-94-C-0001, Internet Infrastructure Protection.
Not approved for public release. Limited distribution only.

<u>This page intentionally left blank.</u>

Key Management for Large Dynamic Groups: One Way Function Trees and Amortized Initialization

By David M. Balenson, Dr. David A. McGrew, and
Dr. Alan T. Sherman

1. Introduction

Efficiently managing cryptographic keys for large, dynamically changing groups is a difficult problem. Every time a member is evicted from a group, the group key must change; it may also be required to change when new members are added. The members of the group must be able to compute a new key efficiently, while arbitrary coalitions of evicted members must not be able to obtain it. Communication costs must also be considered.

Real-time applications, such as secure audio and visual broadcasts, pay TV, secure conferencing, and military command and control, need very fast re-keying so that changes in group membership are not disruptive. To deal with large group sizes (e.g. 100,000 members), we seek solutions whose re-keying operations “scale” well in the sense that time, space, and broadcast requirements of the method grow at most logarithmically in the group size. Key management for these applications should be able to take advantage of efficient broadcast channels, such as radio broadcast and Internet multicast.

We present a new practical algorithm for establishing shared, secret group communications keys in large, dynamic groups. Our algorithm [McG98, Hard97, Bal98], which is based on a novel application of one-way function trees, scales logarithmically in group size. In comparison with previously published methods, our algorithm achieves a new low in the required broadcast size.

2. Previous Work

The broad and challenging problem of establishing keys for large dynamic groups touches upon a large body of previous work, including algorithms for key establishment, group management, authentication in large groups, and multicast security. As for key management, unfortunately very little work has been done on this critical problem in the open literature (e.g. see Schneier [Sch96, pp. 169-187]).

2.1 Key Establishment Algorithms for Large Groups

Prior methods for establishing keys in groups fall roughly into five categories: simple methods that scale linearly in group size, information-theoretic approaches, algorithms based on group Diffie-Hellman key exchange, hybrid methods that trade off information-theoretic security for reduced storage requirements, scalable methods based on hierarchies, and other techniques. Sherman [Bal98, Section 7], Kruus [Kru98], Menezes [Men97], and Just [Jus94] survey some of these methods.

Unfortunately, the information-theoretic approaches [Blu92, Sti96, Chi89] require exponential space to achieve forward secrecy against arbitrarily many colluding evicted members. Although the group Diffie-Hellman methods [Ste96-7, Bur97, Bur94] offer attractive distributed functionality, they suffer from a linear number of expensive public-key operations. Similarly, the hybrid approaches [Fia93, Ber91] scale linearly or worse. As for other techniques (e.g. [Blo90, Gon89]) that do not fall nicely into any of the above categories, we have not found any that provide adequate security.

Therefore, for large groups, the leading candidates are the hierarchical methods, which scale logarithmically in group size. There are two hierarchical methods that do not require trusted internal nodes: the Logical Key Hierarchy (LHK) of Wallner, Harder, and Agee [Wal97, Won97], and the One-way Function Tree (OFT) method of McGrew and Sherman [McG98]. OFT was developed at TIS Labs at Networks Associates, Inc., as part of the DARPA-funded Dynamic Cryptographic Context Management (DCCM) Project [Bal98]; it is the subject of this memo.

The hierarchical methods of Ballardie [Bal96, Bal97] and Harkins [Hark98a] are scalable but require trusted routers.

In addition, the linear Single Key Distribution Center (SKDC) approach is attractive for its simplicity -- at least for relatively small groups (e.g. see [Harn97a-b, Harn98]).

2.2 Managing Large Groups

Managing large groups is important for group communications and for a variety of other applications, including distributed fault-tolerant computing and virtual private networks. Researchers have addressed important group-management issues including defining group membership, supporting distributed fault-tolerant applications, and effecting decentralized dynamic group management.

An important problem in group management is the problem of defining group membership. To support the design of secure, asynchronous, distributed, fault-tolerant systems, Michael Reiter [Rei94] devised a group-membership protocol that tolerates malicious corruption of up to one third of the participants. This protocol is useful in building systems that are robust against limited failures (e.g. hardware failure of some nodes), and that through threshold techniques distribute trust among two or more parties. By contrast, a single group controller is a single point of failure and hence not fault tolerant.

Although group keying is often used to secure group communications, another application of group keying arises in security architectures for distributed fault-tolerant computing. For example, Kenneth Birman [Rei93] and his research group at Cornell have studied how the notion of a secure process group can be used to effect secure, distributed, fault-tolerant computing. Their efforts include the ISIS, Horus [Hor], and Ensemble [Hay98] Systems, which provide a framework and toolkit for developing distributed applications.

Birman [Rod97] and his group have also applied similar ideas to design a virtual private network that can handle network faults, decentralized management, and dynamic membership. Unfortunately, their "solution currently scales [only] to approximately 100 machines" [Rod97, p. 14]. Also, they unwisely claim, that for data confidentiality, "[their] keys are so dynamic and short-lived [changed once a minute] that the approach could be used with a fairly weak cryptographic scheme" [Rod97, p. 1].

Li Gong [Gon96, Keu96] designed and implemented a toolkit called Enclaves for building secure user-level group applications. Enclaves enable users to form virtual private networks on the Internet dynamically. His methods, however, do not scale to large groups. Other recent work in group management includes research by Fenner [Fen97].

2.3 Authentication in Large Groups

There are several ongoing projects to develop infrastructures to support authentication. Among these are the following. The X.509 [Ken93] approach is based on a hierarchical global name space. By contrast, the SDSI/SPKI approaches of Rivest and Lampson [Riv96], and Ellison [Ell97] are based on linked local name spaces. The Secure DNS approach of Eastlake [Eas96] builds on the existing DNS (Domain Name System).

In addition, there is work on batch authentication [Nac94], which provides a way to verify many certificates simultaneously for certificates signed by the same authority under the same signature key.

2.4 Multicast Security

Securing multicast is an active area of research. Some examples of this research are works by Kent [Ken81], Gong and Shachan [Gon94], Ballardie and Crowcroft [Bal95], Deering [Dee98, Dee89], Bagnall [Bag97], Mittra [Mit97], Caronni, et., al. [Car98], and Canetti and Pinkas [Can98], which address issues, requirements, architectures, protocols, and techniques. In addition, the works by Ballardie [Bal96] and Harkins [Hark98a] on key establishment discuss a variety of security problems and solutions for multicast. Securing Mbone [Kum95] broadcasts is one driving application.

3. Group Operations and Their Security Requirements

We envision that the management of group communications keys will take place in a setting in which there will be a communications system, a set of possibly overlapping groups of individuals with common purposes, and individual group members. A systems manager will manage the communications system, and a group manager will manage each group. We envision that groups will comprise a hierarchy of subgroups, with subgroup and organizational managers negotiating on behalf of subgroup members.

3.1 Operations

Associated with each role (system manager, group manager, individual) is a set of operations, minimally including the following operations.

Operations processed by the system manager:

- induct individual into system,
- evict individual from system,
- create group, and
- dissolve group.

Section 5 explains the concept of group induction. In short, the most important results of system (or group) induction are for an individual to establish a base system key known only to the individual and the system (or group) manager, and for the system (or group) manager to check the credentials of the individual.

Operations processed by a group manager:

- add member(s) to group,
- remove member(s) from group,
- evict member(s) from group,
- initiate communications session, and
- terminate communications session.

This document focuses on the crucial operations of adding and deleting members from a group. Note that our OFT method offers some economy of scale when adding or deleting two or more individuals simultaneously. There are two types of membership deletions: a temporary removal (when the individual has not lost his security privileges), and a permanent eviction as the result of loss of security privileges.

Operations requested by individuals:

- request to join group,
- request to leave group,
- request to join session,
- request to leave session, and
- request to return to session.

It is possible that an individual might temporarily lose contact with his group manager --for example, as might happen if an airplane flies out of radio range of his group. Therefore, there must be a way for such a member to re-synchronize key establishment with the group.

3.2 Security Requirements

The primary security requirements for the group operations of adding and deleting members are forward and backward security, as quantified by the degree of forward or backward security [Men97, pp. 528-529]. We say that a method has t-forward security if and only if (iff) no t colluding evictees can read any future communications traffic of the group. Similarly, a method has t-backward security iff no group of t colluding new members can read any previous group traffic. A method has perfect forward (backward) security iff the method has t-forward (t-backward) security for all t.

Backward security is optional. When a new member is added, the group manager may choose to create a new group key, thus denying the new member access to the old key and hence previous traffic. We seek methods, such as OFT, that provide perfect forward security, with the option of perfect backward security.

Carefully note our definitions of the phrases "perfect forward security" and "perfect backward security." Our use of these convenient phrases should not be confused with the different requirement, as explained by Menezes et al. [Men97, p. 496], that compromise of long-term keys does not reveal past session keys. Similarly, our use of these phrases does not necessarily imply information-theoretic "perfectly-secure key distribution" in the sense used by Blundo et al. [Blu92].

Two additional valuable security measures are degree of traceability and degree of disclosure amplification. A method is t-traceable iff more than t colluding members are required to leak plaintext or session keys without being identified if leaked material is discovered. Disclosure amplification refers to the extent of unauthorized disclosure of internal state information (e.g. subgroup keys) caused by the unauthorized disclosure of certain other internal state information.

In addition, when specifying security requirements for particular applications, it is important to understand the security needs and assumptions with regard to any underlying cryptographic primitives (e.g. one-way functions) and with regard to fundamental types of cryptographic strength (e.g. information theoretic, computational, quantum uncertainty).

4. One-Way Function Trees

A One-way Function Tree (OFT) is a binary tree, each node x of which is associated with two cryptographic keys: a node key k_x and a blinded node key k'_x = g(k_x). The blinded node key is computed from the node key using a one-way function g; it is blinded in the sense that a computationally limited adversary cannot find k_x from k'_x.

Although the concept of a one-way function tree is not new (e.g. in 1979, Merkle [Mer79] proposed an authentication system based on a similar idea), our application of this concept is novel.

4.1 Structure of an OFT

A group manager maintains a one-way function tree. Each leaf is associated with a member of the group. The manager uses a symmetric encryption function E to communicate securely with subsets of group members, using unblinded keys as encryption keys as explained below.

Each internal node of the tree has exactly two children. The manager assigns a randomly chosen key to each member, securely communicates this key to the member (using an external secure channel), and sets the node key of the member's leaf to the member's key. The interior node keys are defined by the rule:

k_x = f(g(k_left(x)), g(k_right(x))), (1)

where left(x) and right(x) denotes the left and right child of the node x, respectively. The function g is one-way, and the function f is a "mixing" function (e.g. XOR). McGrew and Sherman [McG98] discuss the properties of f and g in detail. The node key associated with the root of the tree is the group key, which the group can use to communicate with privacy among group members and/or authentication of group membership.

The security of the system depends on the fact that each member's knowledge about the current state of the key tree is limited by the following invariant:

> *OFT Security Invariant* - each member knows the unblinded node keys on the path from its node to the root, and the blinded node keys that are siblings to its path to the root, and no other blinded or unblinded keys.

This invariant is maintained by all operations that add members to the group, and by all operations that delete members from the group.

4.2 Algorithms

The operations of adding and evicting members rely on the communication of new blinded key values, from the manager to all members, after the node key associated with a leaf has changed. To maintain security, each blinded node key must be communicated only to the appropriate subset of members. If the blinded key k'_x changes, then its new value must be communicated to all of the members who store it. These members are associated with the descendants of the sibling of x, and they know the unblinded node key k_s, where s is the sibling of x. To provide the new value of the blinded key to the appropriate set of members, and keeping it from other members, the manager encrypts k'_x with k_s before broadcasting k'_x to the group.

4.2.1 Adding a member

When a new member joins the group, an existing leaf node x is split, creating new nodes left(x) and right(x). The member associated with x becomes associated with left(x), and the new member is associated with right(x). Both members are given new keys. The old member gets a new key because her former sibling knows her old blinded node key and could use this information in collusion with another group member to find an unblinded key that is not on his path to the root. The new values of the blinded node keys that have changed are broadcast securely to the appropriate subgroups, as described in the previous paragraph. The number of blinded keys that must be broadcast to the group is equal to the distance from x to the root plus two. In addition, in a unicast transmission, the new member is given her set of blinded node keys, using the external secure channel. In order to keep the height h of the tree as low as possible, when a new member is added, the leaf closest to the root is split.

4.2.2 Evicting a member

When the member associated with a leaf y is evicted from the group, the member assigned to the sibling of y is reassigned to the parent p of y and given a new leaf key value. If the sibling s of y is the root of a subtree, then p becomes s, moving the subtree closer to the root. In this case, one of the leaves of this subtree is given a new key (so that the evictee no longer knows the blinded key associated with the root of the subtree). The new values of the blinded node keys that have changed are broadcast securely to the appropriate subgroups. The number of keys that must be broadcast is equal to the distance from y to the root.

4.2.3 Initialization

Group initialization is the process through which the group establishes an initial group communications key. For our OFT method, this process involves two steps. First, the group manager broadcasts some information to the group members needed to apply the OFT key-updating procedures. Second, the members compute a shared group communications key, which is needed to begin secure group communications. (Here and throughout, we shall use the verb "broadcast" in the sense of "group broadcast" -- sending a message from the group manager to all members of the group.)

Group initialization is separate from group induction, which is explained in Section 5. During group induction, each member establishes an individual group base key known only by the member and the group manager. Group initialization assumes that each member has already established an individual group base key.

In the first step of OFT group initialization, the manager broadcasts every blinded node key in the OFT to all group members. In this broadcast, each blinded node key is encrypted by the unblinded key of the sibling node, so that only members in the sibling subtree can learn the blinded node key. All members receive the entire broadcast, which consists of a sequence of encrypted blinded node keys. The order in which the keys are broadcast is determined by a postorder traversal of the OFT. This procedure enables the members to associate the keys that they need with the encrypted message parts they receive.

4.3 Properties

In this section we comment briefly on the security, resource usage, and salient characteristic features of The OFT method. For more details, see the paper by McGrew and Sherman [McG98].

The security properties of OFT stem from the system invariant stated in Section 4.1, from the strength of the component one-way function, and from the random selection of leaf keys. In short, OFT provides perfect forward secrecy and optional perfect backward secrecy. Thus, arbitrary coalitions of evicted members cannot directly compute the group communications key nor any unblinded node key.

Evicted members have some information about the key tree but not enough to compute directly any unblinded node key. After a member is evicted, the keys along the path from her node to the root change. After this change, the evictee knows only the blinded keys of the siblings of the nodes along the path from the evictee to the root. These blinded nodes are insufficient to compute directly any unblinded key.

Interestingly, OFT is a centralized, member-contributory method. OFT is centralized in the sense that the group manager plays a special trusted role. OFT is member contributory in the sense that each leaf can contribute entropy to the group communication key.

The hierarchical nature of OFT distributes the computational costs of re-keying among the entire group, so that the manager's computational burden is comparable to that of a group member. Table 1 below summarizes the salient resource usage of adding or deleting a member with OFT in terms of time, memory, number of bits broadcast, and number of random bits needed.

Resource Measure	***Group Member Cost***	***Group Manager Cost***
Time	h	h
Memory	hK	2nK
Bits broadcast	0	hK + h
Random bits generated	0	K

Table 1: Summary of resource usage of adding or deleting a member with OFT. Here, n is the group size, K is the size of a key in bits, and h is the height of the OFT (h = lg n when the tree is balanced). Either the member or the manager could generate the random bits needed at the leaves.

4.4 Implementation Notes and Example

Several important engineering decisions must be made in the implementation of the OFT algorithm: the choice of f and g, the format of broadcasts by the manager, the representation of the tree, and time-space tradeoffs by each member involving how many unblinded ancestor keys to store.

The function g can be based on a cryptographic hash function such as MD5 [Riv92] or SHA-1 [Sha-1]. It is possible that the node keys do not need to be as large as the output size of the underlying function. For example, MD5 has a sixteen-byte output, while DES keys are only seven bytes long. The function g can be constructed from MD5 by discarding some of the output, as is done by S/KEY, so that the node keys (and thus the broadcasts) are smaller.

The function f does not need to be one-way; it needs only to mix its inputs. This fact suggests that f(x,y) = x XOR y is a fast, simple, and effective choice (XOR denotes the bitwise exclusive-or function).

The representation of the tree and the formats of the messages from the group manager are important but routine engineering decisions. For example, the tree could be represented as a record and pointer structure, or as a linear array. Message formats for messages broadcast from the manager could explicitly include node number information to identify which parts of a message correspond to which subtrees, or such topological information could be implicit in the ordering of the message parts.

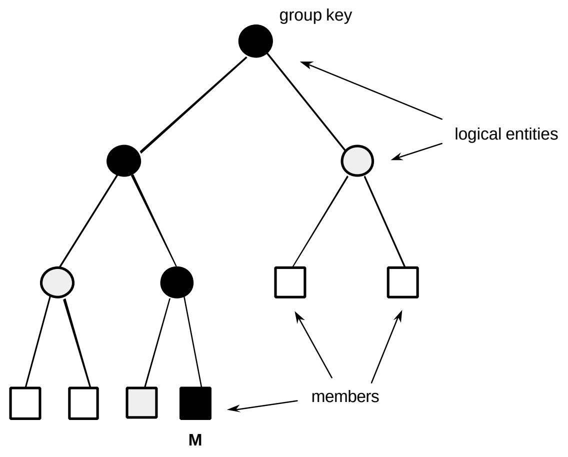

Figure 1: An example of an OFT key tree. The member at the leaf labeled M knows the keys of the solid nodes (including the root key, which is used as the group key), and the blinded keys of the shaded nodes.

4.5 Comparisons with Other Leading Key-Establishment Methods

In this section we briefly compare our OFT algorithm against the two leading competing algorithms for establishing keys in large dynamic groups: the Logical Key Hierarchy (LKH) of Wallner, Harder, and Agee [Wal97], and the Single Key Distribution Center (SKDC). As reviewed in Section 2, these two algorithms and OFT are the main choices available to system engineers who do not wish to trust network routers in large group applications. OFT and LKH are appealing because their computation, broadcast, and memory requirements scale logarithmically with group size. Despite its linear complexity, SKDC is appealing for its simplicity.

As suggested in an observation attributed to Radia Perlman, the add-member operation in LKH can be significantly improved by computing the new group communications key as the result of applying a one-way function to the current group communications key. We shall refer to this refinement of LKH, which maintains perfect backward security, as LKH+. Unfortunately, this refinement applies neither to the delete-member operation nor to the OFT algorithm -- in OFT, this refinement would violate the main system invariant. Fortunately, in many applications, add-member is performed much more frequently than is evict-member. Independently of Perlman, V. Viswanathan [Vis96, pp. 25-26] also suggested a similar constant-time member-parallel idea.

4.5.1 Comparison Criteria

The choice of which key-establishment algorithm should be used can be answered meaningfully only in the context of the requirements of particular applications. To assist engineers in making their choices, we shall summarize some of the major properties of SDKC, LHK, LKH+, and OFT in terms of selected quantitative comparison criteria. These criteria include total delay, number of bits broadcast, number of bits unicast, manager computation, maximum member computation, number of random bits generated, and manager and member memory requirements. Each of these methods provides perfect forward security with the option of perfect backward security.

The four leading candidate methods differ also in terms of required primitives, security semantics, central versus contributory flavor, and resynchronization capabilities (for dealing with group members who temporarily are unable to receive manager broadcasts). For example, SDKC and LKH require encryption functions; LKH+ and OFT require encryption functions and one-way functions. Although none of the authors of any of these methods has provided a formal proof of security, some engineers may have greater confidence in the security of methods with simpler security semantics. Listed in increasing complexity of security semantics, the methods are: SDKC, LKH, LKH+, OFT. With regard to the source of entropy of random bits in common group communication keys, SDKC, LKH, and LKH+ may be viewed as member non-contributory methods because the group manager provides all of the entropy. By contrast, OFT can be used in either a member non-contributory or member-contributory fashion. Most applications have a need to provide a resynchronization capability; some methods may provide some degree of passive member-synchronization.

When asked why she based her LKH method on a keyed encryption function rather than on a faster keyless one-way function (as used in OFT), Wallner [private communication (11/19/97)] explained that her motivating application was a radio communication system. In this application, the available hardware supported a keyed encryption function but not a keyless one-way function. Wallner also remarked (without connection to the choice of encryption function versus one-way function), that in her application, it was very important to conserve battery power. Although these considerations were important to her application, other applications might have different requirements or available primitives; thus the rationale for Wallner's choices do not necessarily apply to other applications.

4.5.2 Quantitative Comparison of SKDC, LKH, LKH+, and OFT

Tables 2 and 3 summarize the time, broadcast, and space requirements of each of the four leading candidate algorithms (SKDC, LKH, LKH+, and OFT). Specifically, for each algorithm and for each of the initialize, add-member, evict-member operations, Table 2 summarizes the total delay, number of bits broadcast, number of bits unicast, manager time, maximum member time, and number of random bits generated. Table 3 summarizes the manager and member memory requirements. For more details, see McGrew and Sherman [McG98].

Initialization

Resource Measure	**SKDC**	**LKH**	**LKH+**	**OFT**
Total delay	n	2n	2n	3n
Number of bits broadcast	NKK	2nK+h	2nK+h	2nK+h
Number of bits unicast	0	0	0	0
Manager computation	n	2n	2n	3n
Max member computation	1	h	h	2h
No. of random bits generated	K	2nK	2nK	nK

Add member

Resource Measure	**SKDC**	**LKH**	**LKH+**	**OFT**
Total delay	n	2h	1	3h
Number of bits broadcast	nK	2hK+ h	h	hK+h
Number of bits unicast	0	0	K	hK
Manager computation	n	2h	1	3h
Max member computation	1	h	1	2h
No. of random bits generated	K	hK	0	K

Evict Member

Resource Measure	**SKDC**	**LKH**	**LKH+**	**OFT**
Total delay	n	2h	2h	3h
Number of bits broadcast	nK+lg n	2hK+h	2hK+h	hK+h
Number of bits unicast	0	0	0	0
Manager computation	n	2h	2h	3h
Max member computation	1	h	h	2h
No. of random bits generated	K	hK	hK	K

Table 2: Summary of time and communication usage of initialization, add-member, and evict-member with the SKDC, LKH, LKH+, and OFT key-establishment methods. Here, n is the group size, K is the size of a key in bits, and h is the height of the key tree (h = lg n when the tree is balanced). Note that while LKH and LKH+ have lower magnitudes of total delay, the units of time for OFT are typically much smaller because keyless one-way functions are much faster than are keyed-encryption functions.

Resource Measure	**SKDC**	**LKH**	**LKH+**	**OFT**
Manager storage	nK	2nK	2nK	2nK
Max member storage	2K	hK	hK	hK

Table 3: Summary of manager and member memory usage in the SKDC, LKH, LKH+, and OFT Key-establishment methods. Here, n is the group size, K is the size of a key in bits, and h is the height of the key tree (h = lg n when the tree is balanced).

4.5.3 Summary

For moderate size groups, the simple SKDC may often be appealing. For very large groups, however, many applications will likely demand a method that scales logarithmically in total delay and member memory usage. For such applications, especially if the add-member is more frequent than the evict-member operation, the LKH+ method looks very attractive for its constant-time add-member and relatively simple security semantics. If for the application it is critical to minimize the number of bits broadcast or the number of random bits generated, or if a member-contributory method is needed, then OFT may be the method of choice.

5. Amortized Group Induction

Before a group can establish an initial group communications key, the members must be inducted (enrolled) into the group. For centralized key establishment methods including OFT, an important objective of this induction step is for each member to establish an individual base key known only to the member and the group manager. The induction process also ensures that each member has the necessary certificates to take advantage of the supporting authentication infrastructure. The main computational goal of induction is to minimize its total delay.

For each group member to establish a separate base key with the Group Manager, the simplest approach is for the manager to engage each member in a separate pairwise authenticated key exchange protocol, such as the Internet Key Exchange (IKE) protocol [Har98, Mau98, Orm]. Unfortunately, this simple approach scales linearly in group size.

In this section we describe a new approach to group induction due to Sherman [She98] that amortizes the relatively high cost of a pairwise key exchange over multiple entries into groups. This amortization saves time when many users become members of two or more groups. We call this new approach "amortized induction." An IBM team [Blu97] independently discovered a similar idea in a network context.

5. 1 Induction Model

Assume there is a universe of N individuals from which G groups are formed, each group having at most n members. Assume further that N is much smaller than Gn; that is, many individuals belong to several groups. It is for this reasonable assumption that amortized induction offers significant savings.

We assume that there is a system that administers multiple groups, each of whose group communications key is established by a key-establishment algorithm such as OFT. Each participant may join one or more group. Each group has a group manager, and there is a system manager who controls induction into the system. For simplicity we shall describe the induction process in terms of a single system manager, though the concept of system manager can be extended to a more general system management function which might be distributed. For each group, each member must establish an individual base key known only to the member and the group manager.

Amortized induction reduces the number of expensive pairwise key exchanges from Gn to N, which can be a significant savings. Amortized induction comes at the cost of Gn one-way function applications by the system manager, but these function evaluations are much faster than pairwise key exchanges. For example, a single application of the MD5 hash function is approximately 1000 times faster than a single ISAKMP/OAKLEY key exchange.

5.2 Induction Algorithm

Whenever an individual first enters the system, the member establishes an individual system base key known only to the individual and the system manager. For example, this step could be carried out using the ISAKMP/OAKLEY Key-Exchange Protocol. Thereafter, whenever the individual joins a group, the individual can establish an individual group base key with the group manager as a one-way function of the individual's system base key, the individual's name, and the group name.

More specifically, let A be the group name. For each member M, the group base key KA_M for M is computed as a one-way function F of M's system base key K_M, the name of M, and the group name. For example,

$$\mathbf{KA\_M = F(K\_M, M, A),} \qquad (2)$$

where F is a publicly known one-way function such as the hash function MD5. The total member delay in this computation is constant.

The function F must satisfy the following properties: it must be easy to compute given its three inputs, and it must be infeasible for anyone to compute the output given only the last two inputs (even under an adaptive chosen plaintext/ciphertext attack).

Whenever a group manager is appointed, the group manager establishes a manager key k_A known only to the group manager and the system manager. For added security this key establishment could be carried out using the same pairwise key-exchange protocol used in the induction process; alternatively, it would be possible to compute manager keys along the lines of Equation 2.

Whenever members of a group need to be inducted, the system manager computes the group base keys and unicasts them to the group manager, encrypted under the manager key k_A. The length of this unicast is linear in the group size. Note that, unlike the SKDC method, no group broadcast is required. Members of different groups can be inducted in parallel.

5.3 An Example of Induction

To illustrate the amortized induction process, consider a toy example in which there are three individuals Q, R, S and two groups A = {Q, R} and B = {S}.

When Member Q is inducted into the system, he establishes a base system key K_Q known only to Q and the system manager. Similarly, Members R and S establish base system keys K_R and K_S, respectively. At the appointment of Group Manager A, Manger A establishes a manager key k_A known only to Manager A and to the system manager. Similarly, Group Manager B shares a manager key k_B with the system manager.

To induct members of Group A, the system manager computes the group base keys KA_Q, and KA_R for Members Q and R, respectively, and sends them to Manager A encrypted under manager key k_A. For example, KA_Q = f(K_Q, Q, A) and KA_R = f(K_R,R,A). In parallel, Members Q and R each computes his own group base key. Members of Group B are similarly inducted.

6.Conclusion and Open Problems

We have presented and analyzed a new practical hierarchical algorithm for establishing shared cryptographic keys for large, dynamically-changing groups. Our algorithm is based on a novel application of One-way Function Trees (OFTs).

Unlike all previously proposed solutions based on information theory, public-key cryptography, hybrid approaches, or a single key distribution center, our OFT algorithm has communication, computation, and storage requirements, which scale logarithmically with group size, for the add or evict operation. Each of the aforementioned methods scale linearly or worse.

In comparison with the only other proposed hierarchical method that does not depend on trusted routers –- the Logical Key Hierarchy (LKH) [Wal97] -- our OFT algorithm reduces by half the number of bits broadcast by the manager per add or evict operation. The user time and space requirements of OFT and LKH are roughly comparable. For many applications, including multicasts, minimizing broadcast size is especially important.

An improvement to the LKH method which we call LKH+ (see Section 4.5), however, offers a significant improvement to the add operation in LKH, reducing the cost of the add-member operation to one one-way function application, a unicast of one key, and no broadcast. This improvement to LKH, together with LKH's relatively simple security semantics, makes LKH+ an attractive choice for many applications.

The OFT method is a centralized algorithm with the option of member contributions to the entropy of the common communications key. Its main advantages are that, in comparison with LKH+, it reduces by a factor of two the number of bits required to be broadcast for each evict-member operation. Our preliminary security analysis of OFT [McG98] raises some interesting questions about the security of function iterates, and that of bottom-up one-way function trees.

It is important to realize that there are significant fundamental limitations to achieving security in large groups -- one might even say that a secure large group is an oxymoron. In most large groups, it is very likely that at least one member is unreliable, untrustworthy, malicious, or careless. Each member knows the common communications key and the plaintext, which is the main commodity being protected. Using multiple communications keys for different subgroups would not enhance security since each member would still have the plaintext. Any member could disclose the plaintext. Consequently, in large groups, it becomes especially important to detect traitors (e.g. through fingerprints and watermarks [Kur98, Sta97, Cho94]) and to limit the loss caused by disclosures (e.g. by rapid evictions and re-keying). Special-purpose, physically secure hardware may play a role in these objectives, by restricting access to communication keys, complicating effective use of compromised keys, and providing unique fingerprints.

Our OFT algorithm offers a practical approach with low broadcast size to manage the demanding key establishment requirements of secure applications for large, dynamic groups.

Acronyms and Abbreviations

DCCM	Dynamic Cryptographic Context Management (a DARPA project [Bal98])
DNS	Domain Name System
GDH	Group Diffie-Hellman (a Group key exchange protocol)
LKH	Logical Key Hierarchy
LKH+	Logical Key Hierarchy, with improved constant-time add-member operation
MSMP	Multicast Security Management Protocol (an NSA/Sparta effort [Harn98])
NAI	Network Associates, Inc.
OFT	One-way Function Tree
SKDC	Single Key Distribution Center
SMG	Secure Multicast Group
TIS	Trusted Information Systems, Inc.
XOR	Bit-wise exclusive-or

Acknowledgments

This work was done as part of the Dynamic Cryptographic Context Management (DCCM) project at TIS Labs at Network Associates, Inc. (formerly the Advanced Research & Engineering (AR&E) Division of Trusted Information Systems (TIS)). Support for the DCCM project was provided by the Defense Advanced Research Project Agency (DARPA) through Rome Laboratory under Contract No. F30602-97-C-0277.

At the time of the work, David McGrew was a cryptography researcher and Alan Sherman was a contractor on sabbatical leave from The University of Maryland, Baltimore County (UMBC). McGrew and Sherman designed and analyzed the OFT algorithm; Sherman devised the amortized induction procedure; and Balenson assisted with project management and final document preparation. In October 1998, McGrew became the head of the Cryptographic Software Development Group at cisco Systems.

We would like to acknowledge contributions by several colleagues at TIS Labs. We are very grateful to Michael V. Harding for significant contributions to the original algorithm and for discussing implementation strategies with us. Dennis K. Branstad, Principal Investigator (former) of the DCCM Project, suggested the problem and provided constructive recommendations and technical guidance. Thanks also to Jay Turner for helpful comments, and to Sharon Osuna and Caroline Scace for editorial suggestions.

References

[Bag97] Bagnall, P., R. Briscoe, and A. Poppitt, "Taxonomy of communication requirements for large-scale multicast applications," Internet Draft (work in progress), draft-ietf-lsma-requirements-01.txt, Internet Engineering Task Force (November 21, 1997).

[Bal95] Ballardie, Tony, and Jon Crowcroft, "Multicast-specific threats and counter-measures," Proceedings of the Internet Society 1995 Symposium on Network and Distributed System Security, February 16-17, 1995, San Diego, California, IEEE Computer Society (1995), 2-14.

[Bal96] Ballardie, A., "Scalable multicast key distribution," Request for Comments (RFC) 1949, Internet Engineering Task Force (May 1996), 18 pages.

[Bal97] Ballardie, A. "Core based tree (CBT) multicast routing architecture," Request for Comments (RFC) 2201, Internet Engineering Task Force (September 1997), 14 pages.

[Bal98] Balenson, David M., Dennis K. Branstad, David A. McGrew, and Alan T. Sherman, "Dynamic cryptographic context management (DCCM): Report #1: Architecture and system design," TIS Report No. 0709, TIS Labs at Network Associates, Inc., Glenwood, MD (June 2, 1998). 121 pages.

[Ber91] Berkovitz, S., "How to broadcast a secret," Advances in Cryptology: Proceedings of Crypto 91, Feigenbaum, ed,. LNCS 576, Springer-Verlag (1991), 535–541.

[Blo90] Bloom, Joel, ed., X9.24-1990, "Financial services retail key management," ANSI X9A3 (March 1990). 84 pages.

[Blu92] Blundo, Carlo, Alfred de Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung, "Perfectly-secure key distribution for dynamic conferences," Advances in Cryptology: Proceedings of Crypto92, E. F. Brickell, ed., LNCS 740, Springer-Verlag (1992), 471–486.

[Blu97] Blumenthal, Uri, Nguyen C. Hien, and Bert Wijnen, "Key derivation for network management applications," IEEE Network (May/June 1997), 26-29.

[Bur94] Burmester, Mike, and Yvo Desmedt, "A secure and efficient conference key distribution system," Advances in Cryptology: Proceedings of Eurocrypt 94, A. De Santis, ed., LNCS 950, Springer-Verlag (1994), 275–286.

[Bur97] Burmester, Mike, and Yvo G. Desmedt, "Efficient and secure conference key distribution," Secure Protocols, M. Lomas, Ed., LNCS 1189, Springer-Verlag (1997), 119–130. [Revised and expanded version of the corresponding Eurocrypt 94 paper by the same authors.]

[Can98] Canetti, R. and B. Pinkas, "A taxonomy of multicast security issues," Internet Draft (work in progress), draft-canetti-secure-multicast-taxonomy-00.txt, Internet Engineering Task Force (May 1998).

[Chi89] Chiou, Guang-Huei, and Wen-Tsuen Chen, "Secure broadcasting using the secure lock," IEEE Transactions on Software Engineering, 15:8 (August 1989), 929–934.

[Cho94] Chor, B., A. Fiat, and M. Naor, "Tracing traitors," Advances in Cryptology: Proceedings of Crypto 94, Y. G. Desmedt, Ed., LNCS 839, Springer Verlag (1994), 257–270.

[Dee89] Deering, S., "Host Extensions for IP Multicasting," Request for Comments (RFC) 1112, Internet Engineering Task Force (August 1989). 17 pages.

[Dee98] Deering, S., D. Estrin, D. Farinacci, M. Handley, A, Helmy, V. Jacobson, C. Liu, P. Sharma, D. Thaler, and L. Wei, "Protocol independent multicast-sparse mode (PIM-SM): Motivation and architecture," Internet Draft (work in progress), draft-ietf-idmr-pim-arch-05.txt, Internet Engineering Task Force (August 4, 1998). 26 pages.

[Ell97] Ellison, C. M., "SPKI Requirements" (February 1997). [For latest version, see http://www.clark.net/pub/cme/html/spki.html]

[Fen97] Fenner, W., "Internet group management protocol, version 2," Request for Comments (RFC) 2236, Internet Engineering Task Force (November 1997). 24 pages.

[Fia93] Fiat, Amos, and Moni Naor, "Broadcast encryption," Advances in Cryptology: Proceedings of Crypto93, D. R. Stinson, ed., LNCS 773, Springer-Verlag (1993), 481–491.

[Gon89] Gong, Li, and David J. Wheeler F. R. S., "A matrix key distribution scheme," Journal of Cryptology, 2:1 (1990), 51–59.

[Gon94] Gong, Li, and N. Shacham, "Elements of trusted multicasting," Proceedings of the IEEE International Conference on Network Protocols, Boston, Massachusetts (October 1994).

[Gon96] Gong, Li, "Enclaves: Enabling secure collaboration over the internet," Proceedings of the Sixth USENIX Unix and Network Security Symposium, San Jose, California (July 1996), 149–159.

[Hard97] Harding, Michael, David A. McGrew, and Alan T. Sherman, "A new key-management algorithm for large dynamic groups," transparencies from talk given by Alan Sherman at NSA (November 19, 1997). 8 pages.

[Hark98a] Harkins, Dan, and Naganand Doraswamy, "A secure, scalable multicast key management protocol (MKMP)," Draft (work in progress), cisco Systems and Bay Networks (March 1998). 20 pages.

[Hark98b] Harkins, D. and D. Carrel, "The Internet key exchange (IKE)," Internet Draft (work in progress), draft-ietf-ipsec-isakmp-oakley-08.txt, Internet Engineering Task Force (June 1998).

[Harn97a] Harney, Hugh, Carl Muckenhirn, and Thomas Rivers, "Group key management protocol (GKMP) architecture," Request for Comments (RFC) 2094, Internet Engineering Task Force (July 1997).

[Harn97b] Harney, Hugh, Carl Muckenhirn, and Thomas Rivers, "Group key management protocol (GKMP) specification," Request for Comments (RFC) 2093, Internet Engineering Task Force (July 1997).

[Harn98] Harney, Hugh, and Eric Harder, "Multicast security management protocol (MSMP): Requirements and policy," Draft (work in progress), SPARTA, Inc. (February 23, 1998). 25 pages.

[Hay98] Hayden, Mark Garland, "The Ensemble system," Ph.D. Dissertation, Department of Computer Science, Cornell University (January 1998). 106 pages.

[Hor] The Horus Project, http://simon.cs.cornell.edu/Info/Projects/HORUS/.

[Jus94] Just, Michael K., "Methods of multi-party cryptographic key establishment," MS Thesis, School of Computer Science, Carleton University (August 9, 1994). 77 pages.

[Ken81] Kent, Stephen T., "Security requirements and protocols for a broadcast scenario," IEEE Transactions on Communications (June 1981).

[Keu96] Keung, S., and L. Gong, "Enclaves in Java: APIs and implementations," Technical Report SRI-CSL-96-07, SRI International, Menlo Park, California (July 1996).

[Kru98] Kruus, Peter, "A survey of multicast security issues and architectures," Proceedings 21st National Information Systems Security Conference, October 5-8, 1998, Arlington, VA, 408-420.

[Kum95] Kumar, Vinay, Mbone Interactive Multimedia on the Internet, New Riders Publishing (Indianapolis, IN, 1995).

[Kur98] Kurosawa, Kaoru, and Yvo Desmedt, "Optimum traitor tracing and asymmetric schemes with arbiter," Draft (work in progress), Spring 98). 13 pages.

[Mau98] Maughan, Douglas, Mark Schertler, Mark Schneider, and Jeff Turner, "Internet security association and key management protocol (ISAKMP)," Internet Draft (work in progress), draft-ietf-ipsec-isakmp-10.txt, Internet Engineering Task Force (July 3, 1998). 86 pages.

[Men97] Menezes, Alfred J., Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, Florida (1997).

[McG98] McGrew, David A., and Alan T. Sherman, "Key establishment in large dynamic groups using one-way function trees," TIS Report No. 0755, TIS Labs at Network Associates, Inc., Glenwood, MD (May 1998). 13 pages.

[Mer79] Merkle, Raph C., "Secrecy, authentication, and public-key cryptosystems," Technical Report No. 1979-1, Information Systems Laboratory, Stanford University, Palo Alto, CA (1979).

[Mit97] Mittra, Suvo, "Iolus: A framework for scalable secure multicasting," Proceedings of the ACM SIGCOMM '97, September 14-18, 1997, Cannes, France. 11 pages.

[Nac94] Naccache, David, David M'Raithi, Serge Vaudenay, and Dan Raphaeli, "Can D.S.A. be improved? Complexity trade-offs with the digital signature standard," Advances in Cryptology: Eurocrypt '94, Alfredo De Santis, Ed., LNCS 950, Springer-Verlag (1994), 77–85.

[Orm] Orman, Hilarie K., "The OAKLEY key determination protocol," Internet Draft (work in progress), draft-ietf-ipsec-oakley-02.txt, Internet Engineering Task Force. 48 pages.

[Rei93] Reiter, Michael, Kenneth Birman, and Robert van Renesse, "A security architecture for fault-tolerant systems," Technical Report TR93-1354, Department of Computer Science, Cornell University (June 1993). 29 pages.

[Rei94] Reiter, Michael K, "A secure group membership protocol," Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, California, May 14-16, 1994, IEEE Press (1994).

[Riv92] Rivest, Ronald L., The MD5 Message-Digest Algorithm, Request for Comments (RFC) 1321, Internet Engineering Task Force (1992).

[Riv96] Rivest, Ronald L., and Butler Lampson, "SDSI: A simple distributed security infrastructure," Version 1.1 (October 2, 1996).

[Rod97] Rodeh, Ohad, Ken Birman, and Mark Hayden, "Dynamic virtual private networks," Technical Report TR97-1654, Department of Computer Science, Cornell University (November 26, 1997). 16 pages.

[Sch96] Schneier, Bruce, Applied Cryptography: Protocols, Algorithms, and Source Code in C, John Wiley & Sons (New York, 1996).

[Sha-1] FIPS Publication 180-1, Secure hash standard, NIST, U.S. Department of Commerce, Washington, D.C. (April 1995).

[She98] Sherman, Alan T., "A new amortized approach to group initialization: Refinements and analysis," TIS Report No. 0754, Trusted Information Systems, Inc. Glenwood, MD (March 26, 1998). 12 pages.

[Sta97] Staddon, Jessica Nicola, "A combinatorial study of communication, storage and traceability in broadcast encryption systems," PhD Dissertation, Dept. of Mathematics, Univ. of California, Berkeley, CA (September 1997). 43 pages.

[Ste96] Steiner, Michael, Gene Tsudik, and Michael Waidner, "Diffie-Hellman key distribution extended to group communication," Proceedings of the 3rd ACM Conference on Computer and Communications Security, March 14–16, 1996. 7 pages.

[Ste97] Steiner, M., G. Tsudik, and M. Waidner, "CLIQUES: A new approach to group key agreement," IBM Research Report RZ 2984 (# 93030) (December 12, 1997). 17 pages.

[Sti96] Stinson, D. R., "On some methods for unconditionally secure distribution and broadcast encryption," unpublished document (November 21, 1996). 35 pages.

[Vis96] Viswanathan, Vaidhyanathan, "Unconditionally secure dynamic conference key distribution," MS Thesis, University of Wisconsin-Milwaukee (December 1996). 28 pages.

[Wal97] Wallner, Debby M., Eric J. Harder, and Ryan C. Agee, "Key management for multicast: Issues and architectures," Internet Draft (work in progress), draft-wallner-key-arch-01.txt, Internet Engineering Task Force (September 15, 1998). 18 pages.

[Won97] Wong, Chung Kei, Mohamed G. Gouda, and Simon S. Lam "Secure group communications using key graphs," Technical Report TR-97-23, Dept. of Computer Science, Univ. of Texas at Austin (July 28, 1997). 24 pages.

Composable Replaceable Security Services for Survivable Distributed Systems: Architecture Report

By Richard Feiertag, Eve Cohen, Jeff Cook,
Timothy Redmond, Jaisook Rho

1. Introduction

The goal of the Composable Replaceable Security Services (CRSS) is to develop a security infrastructure that can support the next generation of survivable distributed systems. The CRSS provides an infrastructure that applications can use in a manner independent of various operating system and networking technologies. Toward this goal, we are developing a collection of distributed security services that embody the properties needed for survivability as well as a framework for composing them. The needed properties for survivability include identification and authentication, confidentiality, integrity, and non-repudiation.

The document describes how the CRSS architecture provides the infrastructure that has the following three properties. First, each service has multiple implementations that can coexist at the same time. For instance, the cryptographic service may have implementations for the RSA encryption as well as for the DES encryption. Second, services are designed to be composable. As systems evolve, the composability allows administrators to update individual services as needed without affecting the performance of the other services. Also, composability allows more complex services to be constructed from a few simple basic services and possibly permits these more complex services to be constructed in different ways using different basic services.

Finally, each service is designed to be a part of a distributed environment. Every service consists of two components: the service framework and the collection of service providers that implement the desired service. The service framework accepts requests from applications or service providers and directs them to appropriate service providers. For example, for the cryptographic service, when an application requests encryption service, the service framework accepts the request and may forward it to the RSA encryption service provider.

Both the service framework and the collection of service providers run in a distributed environment. The service framework can fulfill a service request by invoking a service provider that resides on the local host or a remote host. Similarly, the framework itself is implemented in a redundant distributed manner allowing, for example, the framework on a host to continue operating even if its local copy of the database of service providers is corrupted.

These three properties are essential for survivability. They provide fault-tolerance, scalability, and flexibility. Multiple implementations that are composable can be made fault-tolerant by identifying multiple ways of providing a given service and constructing a mechanism to use an alternative if one way fails. The alternative may be as simple as using a different service provider to fulfill a service request or more complex such as using a different composition of services to fulfill the request. For example, suppose two applications are using a cryptographic service provider (CSP) to communicate and the CSP becomes inoperable. A different CSP or another instance of the same CSP on a different host could be used to secure the communication.

Designing each service so that it can be implemented in a distributed manner allows the infrastructure to be easily scaled. Additional instances of a service provider can be added as the system grows. The scalability is crucial as the number of services grows and as the inter-relationships among them become more complex, and as the number of variable diverse implementations increase.

Finally, reconfigurability and composability satisfy the flexibility requirement. As services evolve, the these two properties allow administrators to reconfigure existing services to facilitate their interactions with new services.

The first design of the CRSS architecture had three layers: the application-specific security service layer, high-level services, and low-level services. Because of the wide-range of applications we wish to support, we thought we needed to develop application-specific security services. However, after categorizing the application-specific services, we have determined that most applications' security needs can be fulfilled by the four high-level security services described below.

The version of the architecture described in this document has two layers of services: the high-level services and low-level services. The high-level services are those services that directly support application programs. There are four such services. The first three services allow applications to specify the following security properties: confidentiality, integrity, and non-repudiation. They also allow specification of the degree of the assurance they provide, ranging from none to strong.

The first high-level service allows applications to connect with each other. Applications can use the connection service to invoke the telnet program across the network. The second high-level service allows applications to send transactions to each other. Applications can use the transaction service to perform electronic commerce and send electronic mail. The third high-level service allows applications to store and retrieve objects. Applications can use this service to retrieve and store files across networks. Finally, the last high-level service allows applications to execute programs securely across the network.

The low-level services are basic capabilities needed to support high-level services. These services consist of the cryptographic service, credential and certificate management service, database management service, trust policy management service, key recovery service and audit service.

This report describes the architecture of the CRSS. This includes a description of each of the services, both high-level and low-level, and a description of the service framework. This report does not address issues such as how failures are detected, how alternative providers and compositions are identified and selected, or how one measures the survivability of a particular configuration of service providers. These will be the subject of subsequent reports.

2. Architecture Overview

In this section, we give an overview of the architecture of the CRSS. The CRSS is designed as a composition of several interoperating services. Each service provides an interface with a well defined functionality. For example, the crypto- graphic service provides the ability to encrypt and decrypt data and other crypto- graphic operations. These service interfaces combine to provide the CRSS functionality.

The CRSS contains two layers of services as shown in Figure 1. The top layer of services is the point of entry for most applications. These high-level services provide the security support that most applications need. This support allows applications to have secure transactions and connections with security properties such as confidentiality, integrity, identification, non-repudiation and authentication.

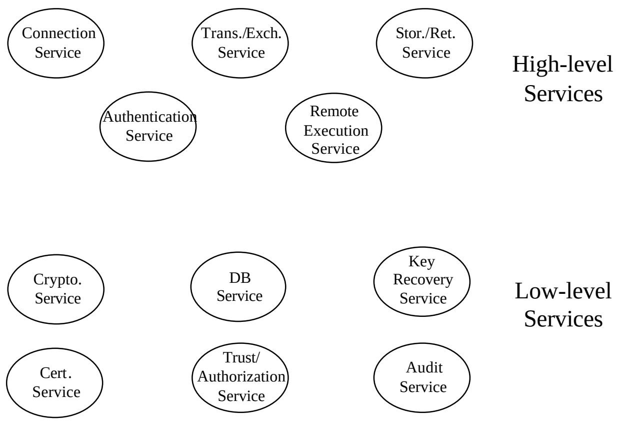

Figure 1 - Security Services Architecture

The bottom level of services provide the mechanisms that the high-level services require. These services provide the high-level services with the ability to do such things as encrypt and decrypt data, examine certificates to determine what they authorize, and store audit records.

The service concept in the CRSS architecture has been designed to support pluggability. The consumer of a service is not restricted to a single implementation of the service interface. Rather, the service interface is implemented by one or more service providers that can be selected dynamically by the consumer or preset by an administrator. For example, the cryptographic service may include a service provider that implements RSA encryption and another service provider that implements DES encryption. These service providers can be added and deleted from a service dynamically.

When a service consumer first desires to use a service, some negotiation must occur to decide which of the service providers performs the right features for the service consumer. By providing a range of service providers that all implement a common interface, the service provides a great deal of choice with regards to choosing the best mechanism to perform a security function. The mechanics of selecting and using a service provider from a given service is described in more detail in a later section. In the remainder of this section, we give an overview of each of these services and an example of how a service provider might be chosen.

2.1 High-Level Services

We now describe each of the high-level services in Figure 1. The connection service is responsible for adding security properties, such as confidentiality and integrity, to a connection between two applications.

An application calls the connection service after the client has obtained an insecure connection. The connection service directs the client through a protocol exchange by which the peer systems set up a common security context for the connection. Depending on the connection service provider chosen, this data exchange may represent, for example, a Diffie-Hellman key exchange or the sending of a kerberos ticket. The security context that is obtained can then be used to impose security on the existing connection. For example, the application can use the security context to encrypt or decrypt data in a manner that is understood by the application's peer.

The transaction/exchange service is responsible for providing security enhancements to data that are used for a single transaction. Examples of the security properties provided by the transaction/exchange service are authentication of the author, integrity, confidentiality and non-repudiation of receipt. This service might be used by an e-mail application or an application that implements electronic money.

When an application desires to add security enhancements to data, it must first set up a security environment. When the application makes the call to create the security environment, it specifies the security properties that it desires. When the security environment is created, the transaction/exchange service informs the application as to what security properties were obtained. The application can then use the security environment to apply security enhancements to data. For example, if the security environment supports confidentiality, then the security environment can be used to encrypt or decrypt data.

The retrieval and storage service is responsible for the secure retrieval and storage of named objects. Such named objects include files and web pages. In general, there are security implications of accessing such named objects. The retrieval and storage service implements the required access controls on named objects. Such access controls can either be based on identity or based on certificate authorization schemes. It is not expected that retrieval and storage of all named objects be accomplished via the retrieval and storage service. Most retrieval/storage mechanisms such as file systems and data bases would likely use other services such as the trust service for implementing security, however, the retrieval and storage service is intended to provide some simple retrieval and storage that has very well understood security properties.

To access a named object, the application opens the named object for a specific access mode. After successfully opening the named object, the application has an object descriptor that it can use to exercise its access to the object. When it is done, the application closes its access to the object.

The remote execution service is responsible for ensuring that executable code transferred between systems can be executed in a safe and secure manner. If potentially malicious code of unknown origin is to be executed then the controls must include some form of sandboxing. Another approach to the remote execution service is to require that the code come with some proof of origin so that it can be executed safely. While remote execution is important, due to its complexity we have deferred design of the remote execution service and it is not included in this report. Many other research efforts are studying the security requirements of remotely executed code and we want our design to reflect results from those studies.

The authentication service is responsible for associating active entities in the system with their identification, authorizations, certificates and credentials. A traditional approach to this problem is to require that a user attempting to use a system must first pass a login challenge. After the user has authenticated himself, all processes started by the user are unforgeably tagged with the user's identity. From there on, determining the authority, certificates and credentials to associate with a process is simply a lookup.

However, the authentication service interface must be rich enough to handle other scenarios. For example, one may wish to authenticate entities other than users such as hosts, programs, or servers. Another possibility is that a user or process may demonstrate its authority by opening a crypto device (which requires a password). The authentication service is designed to incorporate possibilities such as these as well.

2.2 Low-Level Services

The high-level services are intended to be implemented using a set of low-level services as the basis for security. These services, shown in Figure 1, cover the following areas of functionality:

- cryptographic operations,
- key/credential/certificate management,
- trust and authorization policy definition, mediation and enforcement, and
- recording of audit data.

We briefly describe each of the low-level services.

The cryptographic service implements the basic cryptographic operations such as encryption and decryption of data and generation of signatures. In order to use the cryptographic services, the consumer must first create a cryptographic session. In contrast to the high-level services, the cryptographic context creation operations do not take a list of security properties as a parameter. The consumer must have determined that the cryptographic service provider implements the security properties that are desired. The consumer can then use the cryptographic session to encrypt, decrypt, generate digests, sign, verify signature, and generate keys. The cryptographic context should be closed as soon as the desired operations are complete to minimize exposure of key information.

The credential/certificate service provides life cycle support and format-specific manipulation for certificates. This support allows clients to

- create, sign and verify certificates and certificate revocation lists (CRLs),
- view certificates and extract certificate fields,
- import and export certificates to and from other certificate service providers,
- revoke and reinstate certificates, and
- search CRLs.

These calls allow the user to support the entire life cycle of a certificate.

The trust policy service is responsible for determining what authorization is associated with certificates. In this context, a certificate is taken in a broad context. For example, and access control list (ACL) can be considered a certificate and therefore the trust policy service can be used to determine authorization to objects with ACLs. As another example, SPKI defines an algorithm for examining a SPKI certificate to determine what authority can be extended to the entity associated with a certificate. An SPKI trust policy service provider would implement this algorithm.

The key recovery service is responsible for implementing key recovery schemes. Interaction with the key recovery service goes through three phases. First, the consumer must register itself with the key recovery service. This simply initializes key recovery for the consumer. Second, the consumer must enable the key recovery for a particular key blob. This is where key recovery fields are generated and processed. Finally, the consumer may execute a key recovery request to initiate the recovery of a lost key.

The audit service implements a very simple audit log functionality. It has a single call that the consumer of the audit service uses to add some data to the audit log. It is assumed that the data being added represents an audit log entry for some auditable event. This service represents only the interface applications have with audit mechanisms, it is assumed that management of audit logs as well as any analysis of audit logs is defined elsewhere.

3. Framework

3.1 Relationship of Framework to Architecture

The architecture has two parts as shown in Figure 2. The first part consists of the security service providers, shown in Figure 3, to be used by application programs and users. The second part provides the glue that supports invocation of the security providers by application programs, users, and other service providers. We call this part of the architecture the framework as it provides the infrastructure supporting the security services. An important part of the framework is enhancement of the survivability of the security services by enabling replacement of service providers that become inoperable with identical or similar service providers. This replacement can either be directed or automatic.

This potential for replacement provides the redundancy necessary to ensure survivability of the collection of security services in the presence of component failures. Another important aspect of the framework is its support for distribution of the security services. Service providers need not be instantiated on the same host from which they are invoked. The framework allows an application or service provider on one host to invoke a service provider on another host. The fact that the invoked service provider is remote need not be visible to the client. The framework ensures that the remote invocation maintains the security properties requested by the client. The following sections describe the functions provided by the framework, the design of the framework, an overview of provisions for survivability, and an overview of provisions for distribution.

3.2 Framework Functions

The primary purpose of the framework is to allow a client to invoke a service. All of the operations described later in this document are invoked via the framework. The client specifies the operation being invoked and any arguments to that operation and the framework invokes an appropriate service provider. If there is no service provider present to implement the designated service, then the framework returns an error. For most clients, invoking services is the only use they will make of the framework. The framework can automatically locate and invoke a service provider that can fulfill the client's request.

If the service provider is remote, it securely forwards the invocation information to the remote host and, if necessary, receive a reply from the remote host. If the invoked service provider fails to operate, the framework attempts to locate and invoke an alternative service provider that can fulfill the request. The framework continues to seek alternatives until the request is satisfied or no appropriate alternative service providers can be located. For most clients, these processes of location, distribution, and recovery are transparent, i.e., the client need not be aware of them. To the client it is a simple service invocation.

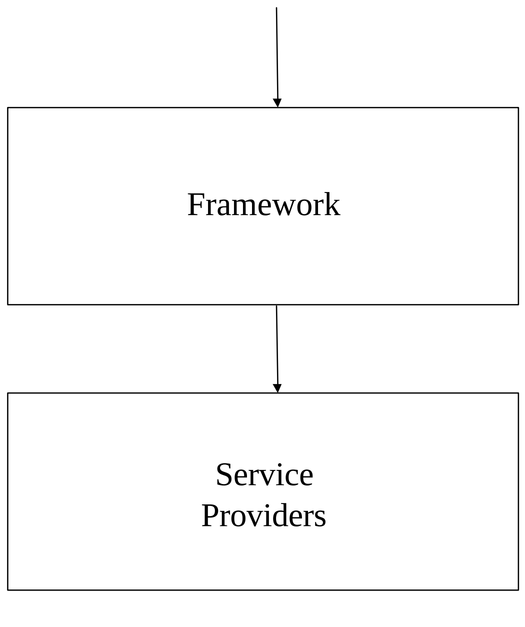

Figure 2 - Services Architecture

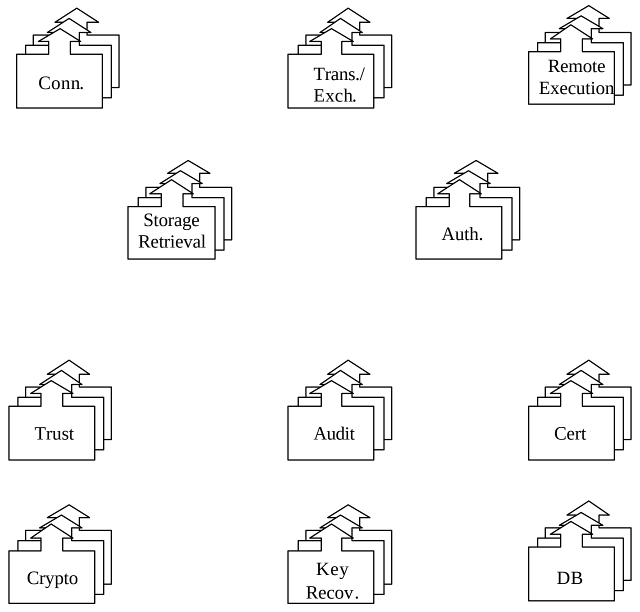

Figure 3 - Service Providers

However, some clients may be able to make use of location, distribution, and recovery of the services they use. To accommodate such clients the framework provides operations to query and advise the framework in actions it takes. The framework provides a registry of service providers. A client can query this registry to locate a particular provider for a particular service. The client can request that invocations of a particular service be fulfilled by a limited set of providers. When that service is invoked by the client then the framework limits itself to locating the restricted set. In this way a client can direct its invocations to particular service providers. A client may want to limit the fulfillment of a service in this way in order to take advantage of special features of certain providers. However, such restriction may adversely effect the reliability of the service by limiting alternatives available to the framework.

Clients may also want to control the distribution of fulfillment of a service. In particular, the framework provides means for a client to restrict the hosts on which providers can run to fulfill service invocations by the client. Also, the client can control the communication mechanisms used to fulfill invocations on remote hosts. When fulfilling a service invocation on a remote host, the framework attempts to use communication mechanisms consistent with the security requirements of the service being invoked. For example, when invoking a remote trust policy provider, the framework communicates with the remote provider using a communication mechanism that preserves the integrity of the data in the invocation. Also, the framework verifies the trustworthiness of the remote provider. Clients can exercise control over the constraints the framework places on remote invocation of service providers. Clients can use this control both to increase the security of remote invocations as well as decrease this security. The ability to decrease security is useful in cases where highly secure providers are unavailable and it is essential to fulfill the service even though security may be compromised.
The framework also provides an interface to service providers to permit providers to register and unregister themselves. As part of registration, the provider provides information about itself including the service it fulfills, its location, and other attributes concerning its operation. Before completing the registration of the provider, the framework may verify properties of the service provider such as its location, the presence of certain operations, and authenticating its origin.

Finally, the framework provides an administrative interface. The administrative interface provides the means to control operation and policy of the framework including such information as what remote hosts it uses for distribution, what properties are required for registration of providers, and what policy it uses for selecting alternative providers. The latter includes determining to what degree the framework allows degradation of service both in security and performance before allowing a service invocation to go unfulfilled.

3.3 Framework Design

The framework consists of three major components as shown in Figure 4. The Provider Registry is a database that maintains information on registered service providers. This includes information on their service, location, attributes, and operation status. Provider Management uses this information to select providers to fulfill service invocations and to select alternatives when providers become inoperative.

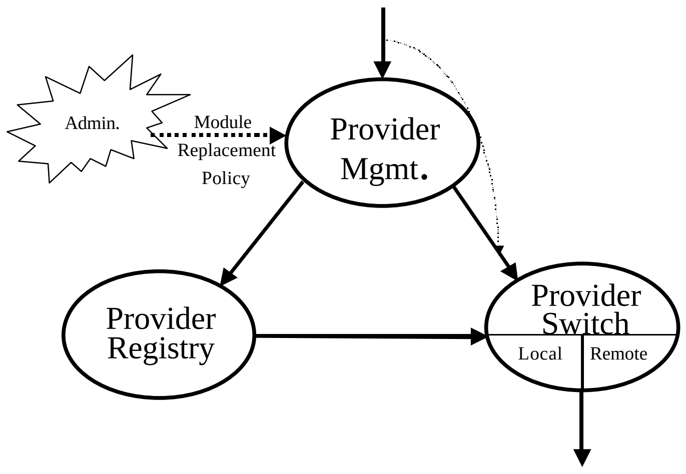

Figure 4 - Framework Design

The Provider Switch performs invocation of service providers. The Switch passes to the provider the information supplied by the client when it invoked the service as well as additional information that the framework needs to pass to the provider. When the invocation is completed, the Switch passes any return information back to the client. The Switch is controlled by Provider Management, i.e., Provider Management tells the Switch which provider to invoke for a given service invocation by a given client. The Switch also monitors the invocation of a provider for unanticipated errors or failure to respond as evidence that a provider is no longer operational. It passes any such evidence to Provider Management. On the basis of such evidence, Provider Management may instruct the Switch to invoke a different provider for subsequent invocations to the service.

Provider Management coordinates the activities of the framework. It accepts all service invocations from clients. In reality these service invocations are passed directly to the Provider Switch (optimizations may allow the service invocations to go directly to the Switch). However, Provider Management manages the Switch, directing it as to which providers are invoked. Provider Management also manages the Registry, accepting registration requests from providers and directing the Registry to incorporate or remove providers in its data base. Provider Management also determines which provider is invoked to fulfill a service request by a particular client. Clients may provide input to Provider Management to influence this decision, but Provider Management makes the decisions and directs the Switch accordingly. Provider Management accepts direction via its administrative interface as to policy for selecting a provider. This information is also used to select alternatives when providers become nonoperational. Providers may also be replaced for other reasons such as the registration of a new provider that may be better suited to fulfill a particular server for a given client.

3.4 Framework Distribution

The framework is intended to operate in a distributed environment. That is, it operates in a coordinated fashion on multiple interconnected hosts. This allows a client running on one host to have a service invocation fulfilled by a provider running on a different host. Some part of the framework must be present in each participating host (for example, the Provider Switch must be present on each host but the registry may not have to be present on each host). The Provider Registry is a distributed data base within the framework. Provider Management is a distributed service that can select from both local and remote providers. Reliability considerations dictate to what extent framework components such as the Registry are present on each host.

The Provider Switch is responsible for communication between the framework on each host. The Provider Switch uses either the Secure Connection or Secure Transaction service to establish secure communication between hosts. Whenever the Provider Switch is used to fulfill a service invocation and the provider is remote, the switch sends the information to the Switch on the remote host which, in turn, invokes the desired provider. The logical path for both local and remote service invocations is illustrated in Figure 5 and Figure 6.

Figure 5 shows a local invocation. The first time a client invokes a particular service is illustrated in Path 1. The Provider Manager selects a provider to fulfill the service invocation by consulting the Registry and forwards the information to the Provider Switch. The Provider Switch then invokes the selected provider. The Switch retains the association between the client and the provider so that in subsequent invocations, the Provider Manager can essentially be bypassed. However, the Provider Manager may at any time direct the Switch to use a different provider for the client. In the case where the provider is remote as shown in Figure 6, instead of invoking the provider, the Switch sends the request to the Switch on the remote host where the provider resides using the Secure Connection or Secure Transaction Service. The remote Provider Switch then makes the invocation of the provider.

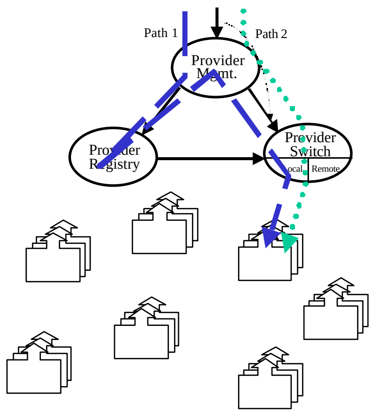

Figure 5 - Local service provider invocation

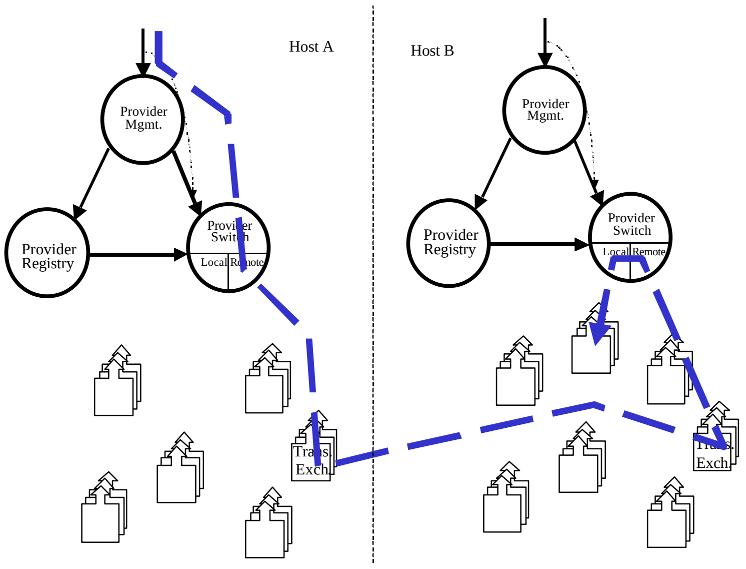

Figure 6 - Remote service provider invocation

Although not specifically indicated on the diagram, the invocation by the Switch of the Secure Connection or Secure Transaction Service is similar to any invocation of the service by a client. This means that the Switch calls the Provider Manager that decides which provider to select to fulfill the service and then forwards the call back to the Switch to actually invoke the provider. When calling the Provider Manager to invoke the Secure Transaction Service, the Switch must request a local provider to avoid an unbounded recursion. Calls between components of the framework are handled in a similar manner. For example, if the instance of the Provider Registry on one host needs to invoke the Registry on another host, it is treated as a service invocation with only one provider, namely the Registry on the latter host. This allows the same mechanisms that assure survivability of security services invoked by clients to be applied to internal invocations, i.e., invocations by the framework itself.

4. Simple Example Application

In this section we describe an example of how an application can utilize the security features of the CRSS in a fault tolerant manner. We describe how the application, e-mail in this case, uses the CRSS in the absence of a fault, then we a fault into the scenario and show how the application and the CRSS responds.

The e-mail application uses the transaction/exchange service in order to obtain security enhancements such as non-repudiation, authentication, confidentiality and integrity. For simplicity of exposition, we restrict our attention to the send e-mail operation and only consider authentication, confidentiality and integrity enhancements.

The transaction/exchange service provides applications with the option of using a single call to provide security enhancements to data. To use this option when an e-mail application is ready to send an e-mail message, it must specify the desired levels of security properties. For example, the application could specify that the message have strong confidentiality and a medium level of integrity. The e-mail application invokes the service framework specifying the transaction/exchange service. The framework selects a transaction/exchange service provider that can perform the requested security enhancements and invokes this provider. If for whatever reason, the selected service provider cannot perform the service, then the framework redirects the request to the next qualified service provider. As part of its function, the framework may have to clean up after the failed service provider.

We start by illustrating the case where no fault occurs. In this case, the framework must select a service provider to fulfill the service request. In this example, this selection will be driven primarily by the available credentials for the sender and recipient of the message. The credentials are the mean by which the sender and receiver authenticate each other and protect the data in transit. The framework must select a service provider that can use the available credentials. Also, the framework must be cognizant of the service providers that are available to the recipient in order to select a service provider that uses credentials that the recipient can use. The instance of the framework on the sender’s host may need to communicate with the instance of the framework on the recipients host in order to select an appropriate service provider. Then, the framework passes the credentials to the transaction/exchange service provider that uses them to provide the desired service.

Now consider what happens when the chosen service provider cannot perform the desired service. The framework, as in previous case, invokes a provider, but the provider fails to provide the requested service for some reason (e.g., it has insufficient resources such as memory to fulfill the request or the credential is no longer valid). In this case, the framework must first recover from this failure in an orderly manner, by effectively restoring its state to that which existed before it invoked the failed provider. Then, the framework must find another provider to attempt to fulfill the request. It continues to recover from failures and try alternative providers until either the request is successfully fulfilled or the set of appropriate providers is exhausted. In the latter case, the framework returns an error to the email application, stating that the request could not be serviced.

5. Low-level Services

The pluggable low-level security services provided by CRSS are listed below.

Low-level Service	Description
Cryptographic	cryptographic service providers
Credential and Certificate Management	syntactic credential and certificate management services
Database	database (local and/or remote storage) services
Trust Policy Management	trust policy management services (certificate semantics)
Key Recovery	key recovery services
Audit	audit services

The low-level services for CRSS have been modeled after the low-level services provided by Intel's Common Data Security Architecture (CDSA) specification, and we intend to use the CDSA services model and API where possible. By using an emerging standard such as CDSA we expect that there will be a large number of service providers for use in CRSS. This will both allow us to make use of the work of others and lead to a more survivable system. The CDSA contains a Common Security Services Manager (CSSM) that manages a set of low-level services. The low-level services of CRSS that are currently supported by CSSM are: Cryptographic, Credential and Certificate Management, Database, Trust Policy Management, and Key Recovery. An Audit Service is rumored to be in the works, but has not yet been made public.

Unfortunately, CDSA is a specification in transition. The newest specification is version 2.0, published in October 1997. The latest specification for which software exists is version 1.2, published in March 1997, and whose software was not available until November or December 1997. The 2.0 specification is much cleaner, better organized, and more capable than 1.2, but, since software is not currently available for 2.0, for CRSS we are initially targeting CDSA 1.2 with the intention to move on to 2.0 when software becomes available. As such, when CRSS services correspond to CSSM services, we present the APIs for both CDSA 1.2 and 2.0.

5.1 Cryptographic Services

The cryptographic services of CRSS are modeled directly after the cryptographic services of CDSA. Low-level cryptographic services are provided by Cryptographic Service Providers (CSPs) that plug into CDSA's Service Provider Interface (SPI). CSPs are add-in modules that may be implemented in software, hardware, or a combination thereof. CSPs perform basic cryptographic operations such as digital signature and verification, encryption and decryption, key and key pair generation, and so forth. A complete list of the operations that may be performed by a CSP are listed below. An individual CSP may choose to implement a subset of the possible operations, and may implement additional vendor-specific operations by using the PassThrough functionality provided for extensibility.

An application may make queries to determine what CSPs are installed and what services they provide. (CDSA has provisions for registration and integrity checking of CSPs to prevent masquerading.) Calls made to a CSP to perform cryptographic algorithms occur within a framework called a session, which is established and terminated by an application. A cryptographic context is created prior to starting a session and is deleted as soon as possible after completion. This contextual information is not persistent and is not saved permanently in a file or database. To protect access to CSP services, CSPs optionally support a password-based login/logout sequence, where users may change their passwords as deemed necessary, and support may even be provided for operations to be performed by privileged CSP administrators.

Secure storage of private keys is always the responsibility of a CSP, which may optionally assume responsibility for the secure storage of objects of other types, such as symmetric keys and certificates. For storage of private keys, a CSP may choose to use the services of a Data Storage Library module within the CSSM framework (if the module provides secure storage) or may use an approach that is internal to the CSP. Access to other persistent objects managed by the CSP are performed using CSSM's Data Storage Library APIs.

Cryptographic operations come in two types -- a single call to perform an operation and a staged method of performing the operation. For the staged method, there is an initialization call followed by one or more update calls, and ending with a completion (final) call. For most staged cryptographic operations, the final result is available after the function completes its execution, the exception being encryption/decryption where each update call generates a portion of the result.

5.2 Credential and Certificate Management Services

The credential and certificate management services provided by CRSS are modeled directly after those provided by CDSA. These services are provided by Certificate Library (CL) modules that plug into CDSA's Certificate Library Interface (CLI). The primary purpose of a Certificate Library is to perform syntactic manipulations on a specific certificate format and its associated Certificate Revocation List (CRL) format. These manipulations include the complete life cycle of a certificate and the keys associated with the certificate. Certificates and CRLs are related by the life cycle model and by the data formats used to represent them. As such, these objects should be manipulated by a single, cohesive library.

Certificate Libraries manipulate memory-based objects only. It is the responsibility of the application and/or the trust policy module to use data storage add-in modules to make objects persistent, when appropriate.

5.2.1 Certificate Life Cycle

A Certificate Library provides life cycle support and format-specific manipulation for certificates. It permits applications and other modules to create, sign, verify, revoke, renew, and recover certificates without requiring knowledge of certificate and CRL format and encodings.

The first step in the life cycle process is registration, during which the authenticity of a user's identity is verified. This may require manual procedures, depending on the Security Policy in force. After registration, keying material is generated and certificates are created, issued to the user, and then backed up if appropriate, after which the active phase of the certificate life cycle begins. The active phase includes:

- *retrieval*: retrieve a certificate from a remote repository
- *verification*: verify the validity dates and signature(s) on a certificate and its revocation status
- *revocation*: assert that a previously-legitimate certificate is no longer valid
- *recovery*: recover a key when the user has forgotten his password
- *update*: issue a new certificate when one will expire soon

5.3 Database Services

The database services provided by CRSS are modeled directly after those provided by CDSA. These services are provided by Data Storage Library (DL) modules that plug into CDSA's Data Library Interface (DLI). The primary purpose of a data storage library is to provide persistent storage of security-related objects such as certificates, CRLs, keys, and policy objects. A DL module is responsible for the creation and accessibility of one or more data stores. A single DL module can be tightly coupled to a CL and/or a TP module, or can be independent of all other module types. A single data store can contain a single object type in one format, a single object type in multiple formats, or multiple object types. The persistent repository can be local or remote.

The abstract data model defined by the DL APIs partitions all values stored in a data record into two categories; one or more mutable attributes and one opaque data object. The attribute values can be directly manipulated by the application and the DL module. Values stored within the opaque data object must be accessed using parsing functions. A DL module that stores certificates can, but should not, interpret the format of those certificates. A set of parsing functions such as those defined in a CL module can be used to parse the opaque certificate object.

5.4 Trust Policy Management Services

The Trust Policy Management Services of CRSS are modeled directly after the Trust Policy Services of CDSA, which includes Authorization Services. Trust Policy Management Services associate semantics with certificates and are provided by Trust Policy (TP) modules that plug into CDSA's Trust Policy Interface (TPI). The primary purpose of a Trust Policy module is to answer the following question:

Is this certificate authorized for this operation in this trust domain?

Applications are executed in a trust domain. For example, executing an installation program at the office takes place within a difference trust domain than performing the same action on a home computer. In the former case, the trust domain is corporate and may require extensive credentials, whereas in the latter case, the trust domain is personal and may only require a credential that establishes the user as a known entity on the local system.

The general CSSM trust model defines a set of basic trust objects that most trust policies use to model their trust domain and the policies over that domain. These basic trust objects include:

- policies,
- certificates,
- defined sources of trust,
- certificate revocation lists,
- application-specific actions, and
- evidence.

Policies define the credentials required for authorization to perform an action on another object. Certificates are the basic credentials representing a trust relationship among a set of two or more parties.

Evaluation of trust depends on relationships among certificates. Certificate chains represent hierarchical trust, where a root authority is the source of trust. Entities attain a level of trust based on their relationship to the root authority. Certificate graphs represent an introducer model of trust, where the number and strength of endorsers increases the level of trust in an entity. In both models, the trust domain can defined accepted sources of trust. In contrast to certificates, certificate revocation lists represent sources of distrust. Trust policies may consult these lists during the certificate verification process.

Trust evaluation can be performed with respect to a specific action the bearer wishes to perform, or with respect to a policy, or with respect to the application domain in general. In the latter case, the action is understood to be either one specific action, or any and all actions in the domain.

When verifying trust, a Trust Policy Module (TPM), which is an encoding of a specific trust policy, processes a group of certificates. The result of verification is a list of evidence, which forms an audit trail of the process. The evidence may be a list of verified attribute values that were contained in the certificates, or the entire set of certificates, or some other information that serves as evidence.

5.5 Key Recovery Services

This section describes those key recovery services provided as an elective module by CDSA 2.0. Key recovery mechanisms facilitate the retrieval of cryptographic keys by authorized parties.

Key Recovery Scenarios

The CDSA Key Recovery Module supports three basic scenarios motivating the use of key recovery mechanisms:

- Individual key recovery: an individual user may need to recover a lost or corrupted key.
- Enterprise key recovery: a corporation may wish to recover keys used by its employees in the case where the employee is unavailable or unwilling to provide them.
- Law enforcement key recovery: key recovery mechanisms may be used by law enforcement bodies to access the contents of confidential communications or stored data in the interests of law enforcement or national security.

The CSSM enforces both the applicable enterprise and law enforcement policies on all cryptographic operations.

Key Recovery Mechanism Types

There are two types of key recovery mechanisms supported by the CDSA Key Recovery Module:

- Key escrow: a trusted party holds the cryptographic keys in question (or portions thereof.)
- Key encapsulation: a cryptographically wrapped form of the key in question is provided to those requiring key recovery. The encapsulated key may be unwrapped only by a trusted party.

5.6 Audit Services

The CDSA currently contains no specification for audit services. However, such services are often mentioned in the context of elective services. [CSSM_EMM_2.0]. For CRSS, we provide audit services via an Audit Library (AL) module that plugs into an Audit Library Interface (ALI). The purpose of an audit library is to provide a repository for a time-stamped sequence of audit events. An

Audit Library (AL) module is responsible for collecting and logging the audit events generated by other CRSS services or by applications that make use of CRSS. We take the approach that it is infeasible to determine which events of an application program or CRSS service are auditible, and, instead, let the application or service choose to audit those events deemed significant. The AL timestamps each audit event received and logs it to a permanent, protected audit repository. Timestamping permits the AL to maintain the proper sequencing of audit events. As a first cut, we intend for the ALI to provide only the single interface that simply timestamps and records audit events. The initial ALI does not have operations for opening, closing, and analyzing audit logs. We assume that all audit analysis is performed off-line.

6. High-Level Services

The high-level security services provided by CRSS are listed in the table below. These services may be invoked by applications, service providers, or the framework.

High-level Service	Description
Authentication	credential management
Secure Connection	securing dynamically established communication between two parties
Secure Transaction/Exchange	file or message protection
Secure Retrieval and Storage	named-object protection and access
Secure Execution	securing local execution of executable objects

6.1 The Authentication Service

The authentication service extends the trust policy management by managing users' credentials. It provides an administrative interface for maintaining the credential database; in addition, it allows other high-level CRSS services, such as the Secure Connection Service, to obtain credentials associated with a user or a role.

There exists a wide variety of methods by which authentication and authorization can be provided [CRYPTO]. A conventional method is to use passwords to authenticate users. Once a user is authenticated to the system, any processes spawned by the user is marked as belonging to that user. When the process initiates an action requiring a key, the system can use the user identification associated with the process to determine if the process is allowed to use the key. A variation of this approach is to have a system configured in such a way that any user sitting at a certain terminal is given a certain authority. Another approach is to use a crypto device to give certain users access to crypto operations. Such a device does not give out keys but instead performs crypto operations for an active entity that can provide the needed personal identification number (PIN). In this approach, the system may not have any identification or authorization information associated with a particular active entity. The authorization is implicit in the active entity's ability to execute certain crypto operations requiring keys.

In the following interface description, the *caller* is the active entity invoking the interface call. The *requester* is the active entity making the original request. In most cases, the requester is a user or an application program. When the requester is the latter, then it must be running on behalf of a user or role.

The authentication service provides two interfaces. The first interface is the administrative interface for creating or importing new credentials and revoking existing credentials. This interface is provided only to those users who are authorized to access the credential database. Whenever a requester wishes to access the credential database, the authentication service checks that the requester has presented a valid credential to perform the operation. The second interface is a programming interface for obtaining access to the credentials of a user or a role. Because the presentation of a credential authenticates a user or a role, the CRSS does not export actual credentials to the application program interface. Instead, application programs must use those interface calls that return credential handles. Other high-level CRSS services may invoke those calls that return credentials and use them on behalf of the application programs.

6.2 The Secure Connection Service

A connection is an agreement about how to communicate that is dynamically established between two parties. Two parties that wish to communicate start with an initial data exchange to define the connection. As a result of this data exchange, both parties can store common information about the connection such as the communication path, quality of service information and security information. This data exchange is known as opening the connection. This information store is then consulted whenever the two parties wish to communicate.

The purpose of the connection service is to take an existing connection and add security services [TLS]. The service does this by defining a data exchange that should be implemented using the existing connection. The approach is as follows. The application opens an insecure connection. One side of this insecure connection, which we call the initiator, asks the connection service for some data to pass to the other side of the connection, which we call the acceptor. The initiator then uses the existing connection to pass the data to the acceptor. This represents the first step in a protocol for implementing a secure connection. The acceptor passes this data to its connection service in order to obtain some data to use to reply to the initiator. The acceptor's connection service returns some data to the acceptor that can then be passed to the initiator. This protocol continues until either or both of the acceptor or the initiator gets an indication from their connection service that the exchange is complete. We are basing this service on that provided by the Generic Security Service Application Program Interface [GSS-API].

The goal of this protocol is the generation of a security context for the initiator and the acceptor. This security context defines an agreement between the initiator and the acceptor that allows them to implement security services on their shared connection. For example, the security context may include a secret key shared between the initiator and the acceptor that allows them to encrypt data passed over the connection. The Diffie-Hellman exchange is an example of a protocol that the connection service could implement that would generate such a shared secret key.

6.3 Secure Transaction/Exchange Service

The purpose of the transaction/exchange service is to service applications requiring protection of a unit of data, e.g., a file or message. Since the protection provided is independent of any concurrent contact with the eventual recipient of the protected data, this service would be useful to such applications as secure electronic mail. We are basing this service on that provided by the Independent Data Unit Protection Generic Security Service Application Program Interface [IDUP-GSS-API], [GSS-API].

The protections provided via the transaction/exchange service interface include the following:

- data origin authentication with data integrity,
- data confidentiality with data integrity, and
- support for non-repudiation services.

The transaction/exchange service has interfaces for the following types of functionality:

- establishing and abolishing the security environment,
- providing the following types of security protection enhancement to the data:
    - signature and encryption protection and unprotection and
    - non-repudiation evidence support, and
- establishing an environment and protecting the data in a single step.

6.4 The Secure Retrieval and Storage Service

The secure object retrieval and storage service provides applications with the ability to access objects by name. Access to the objects is controlled by an access control policy. Therefore, all calls that require an access control check must also take a handle for the credentials of the caller.

6.5 Secure Execution

The purpose of the secure execution service is to provide a mechanism for retrieving a remote object, for example a Java applet, which includes instructions for "execution" of that object; and for insuring that the resulting local execution is "secure". Examples of two very different mechanisms for insuring security might be

- a secure environment for the local execution of the object
- the use of a digital signature to indicate that the object is safe to execute

The design of the secure execution service has been deferred to a subsequent report.

7. Conclusion and Plans

The architecture described above provides a basis for constructing a collection of services that can fulfill the security needs of a wide variety of applications. The architecture also provides a means for maintaining a high degree of survivability for this collection of services. The designs for the specific services given above are initial proposals and are subject to refinement as the development of the CRSS proceeds. In designing these services we have tried to make use of prior work by basing the designs on CDSA, GSSAPI, and GSSAPI-IDUP. We also anticipate that we will be able to use service providers developed for CDSA and possibly portions of the CDSA CSSM in the implementation of the CRSS.

The architecture is only the first step in the development of CRSS. Subsequent steps include the following:

- design of selection algorithms for choosing service providers and identification of the data necessary to support the selection,
- design of techniques and algorithms for replacing service providers in the presence of failures,
- detailed design and specification of the framework,
- detailed specification of the service APIs,
- implementation of the above,
- techniques and tools for analysis of the survivability of the CRSS.

These will be described in subsequent reports.

References

References for CDSA 1.2

[CDSA_1.2] Common Data Security Architecture Specification, Draft Release 1.2, Intel, March, 1997.
[CSSM_API_1.2] Common Security Services Manager Application Programming Interface (API), Draft for Release 1.2, Intel, March, 1997.
[CSSM_SPI_1.2] Common Security Services Manager Cryptographic Service Provider Interface (SPI) Specification, Draft for Release 1.2, Intel, March, 1997.
[CSSM_CLI_1.2] Common Security Services Manager Certificate Library Interface (CLI) Specification, Draft for Release 1.2, Intel, March, 1997.
[CSSM_DLI_1.2] Common Security Services Manager Data Storage Library Interface (DLI) Specification, Draft for Release 1.2, Intel, March, 1997.
[CSSM_TPI_1.2] Common Security Services Manager Trust Policy Interface (TPI) Specification, Draft for Release 1.2, Intel, March, 1997.

References for CDSA 2.0

[CDSA_2.0] Common Data Security Architecture Specification, Draft Release 2.0, version 1.0, The Open Group, October, 1997
[CSSM_API_2.0] CSSM Application Programming Interface, Draft Release 2.0, version 1.0, The Open Group, October, 1997.
[CSSM_SPI_2.0] Cryptographic Service Provider Interface Specification, Draft Release 2.0, version 1.0, The Open Group, October, 1997.
[CSSM_CLI_2.0] Certificate Library Interface Specification, Draft Release 2.0, version 1.0, The Open Group, October, 1997.
[CSSM_DLI_2.0] Data Storage Library Interface Specification, Draft Release 2.0, version 1.0, The Open Group, October, 1997.
[CSSM_TPI_2.0] Trust Policy Interface Specification, Draft Release 2.0, version 1.0, The Open Group, October, 1997.
[CSSM_EMM_2.0] CSSM Elective Module Management, Draft Release 2.0, version 1.0, The Open Group, October, 1997.
[CSSM_KR_API_2.0] Key Recovery Application Programming Interface Specification, Draft Release 2.0, version 1.0, The Open Group, October, 1997.
[CSSM_KRI_2.0] Key Recovery Service Provider Interface Specification, Draft Release 2.0, version 1.0, The Open Group, October, 1997.

Other References

[CRYPTO] *Applied Cryptography, Second Edition,* Bruce Schneier, John Wiley and Sons, Inc., 1996.
[GSS-API] Generic Security Service Application Program Interface, Version 2 , J. Linn, OpenVision Technologies, January 1997.
[IDUP-GSS-API] *Independent Data Unit Protection Generic Security Service Application Program Interface (IDUP-GSS-API)*, D. Adams, Entrust Technologies, November 1997.
[KERBEROS] *The Kerberos Network Authentication Service (Version 5),* Internet RFC-1510, September, 1993.
[TLS] The TLS Protocol, Version 1.0, Tim Dierks and Christopher Allen, Consensus Development, November, 1997.

Domain and Type Enforcement Firewalls

By Karen A. Oostendorp, Lee Badger, Christopher D. Vance, Wayne G. Morrison, David L. Sherman, and Daniel F. Sterne

1. Introduction

Internet-connected organizations often employ an Internet firewall to mitigate risks of system penetration, data theft, data destruction, and other security breaches. Conventional Internet firewalls, however, impose an overly simple inside-vs-outside model of security that is incompatible with many business practices that require extending *limited trust* to external entities, for example, suppliers, bankers, accountants, advisors, consultants, partners, customers, and allies. Additionally, firewall security perimeters are somewhat weak: they provide no protection from inside attacks and do not protect sensitive data, which can be exported by tunneling through permitted protocols. These problems will likely become more acute in the next few years as applets [9], mobile agents [19], and object frameworks [13] provide opportunities for malicious programs to execute behind firewall security perimeters.

We believe that, while firewall security perimeters are helpful, a fundamentally stronger mechanism will be required to protect "sacrificial lamb" network servers, to run network clients in environments that control damage inflicted by malicious servers (and applets), and to provide a basis for controlled sharing of information between organizations. A number of computer operating system security controls [2, 4, 3, 10, 7] have been proposed, but most of these techniques either map poorly to commercial requirements or impose excessive administrative overheads. Domain and Type Enforcement (DTE) [1] is a relatively recent operating system access control mechanism that holds promise to provide needed security flexibility and strength while controlling administrative costs. We believe that a fundamental question for practical commercial security is whether strong-but-flexible access controls, such as DTE, can be combined with firewall defenses in a way that preserves the practicality of firewalls while substantially improving security.

This paper reports on our experience with a DTE-enhanced firewall prototype. Our preliminary results indicate that DTE access controls *can* be integrated with firewalls to cost-effectively control resources shared through firewalls and to protect enterprise resources from possibly malicious insider programs. After reviewing the primary concepts of DTE, this paper presents the design of a DTE firewall and the mechanisms it uses to control imported and exported services. This paper next presents our informal evaluation of DTE firewall security, functionality, compatibility, and performance characteristics. Finally, this paper reviews related work, future directions, and presents conclusions.

2. DTE Review

DTE [1, 18, 15] is an enhanced form of type enforcement [4, 12, 14], which is a table-oriented mandatory access control mechanism. DTE has three main benefits over type enforcement. First, the security policy is specified in a high level language that reduces the burden of expressing, verifying, and maintaining security rules. Second, security attributes on objects are implicit, thereby allowing a file hierarchy to be typed concisely. Finally, DTE provides mechanisms for backward compatibility with existing software and with systems not running DTE.

As with a number of other access control mechanisms [2, 3, 4, 7, 11], DTE considers a system to be logically split into two categories: active entities (usually processes) and passive entities (e.g., files or network packets). A type is associated with each passive entity, or object; a domain and a DTE-protected user identifier (unchangeable even by root) is associated with each active entity, or subject. Using a language-based specification, DTE expresses allowed interactions between subjects and objects. Access control decisions are made by consulting a DTE database consisting of the "compiled" specification to determine if the domain has the requested access (e.g., read or write) to the object. DTE also expresses the allowed interactions between different subjects. Access control decisions consult the DTE database to determine if subject A has the requested access (e.g., execute or kill) to subject B. A DTE system's security posture is completely determined by its policy specification.

To extend DTE protection across networks, DTE treats each network packet as an object with three associated attributes (carried in the IP option space of each datagram): the DTE type of the information, the domain of the source process (source domain), and the DTE-protected uid of the source process. A process can read or write a message object only if the process's domain has the appropriate access to the type of the message. When a message originates from a non-DTE system, domain and type information are assigned by the receiving DTE system. Similarly, a DTE system ensures that messages are not sent to a non-DTE system unless its associated domain can read the messages.

In our prototype, UNIX[2] kernel changes to enforce DTE mediation are localized to a relatively small subsystem; all system calls that represent process accesses to other processes or to objects are passed through this subsystem.[3] During the initialization phase of a DTE kernel (i.e., at boot-time), the DTE subsystem reads a security specification written in the DTE Language (DTEL) from the boot device, parses the specification into access control data structures, and then mediates the system calls of all processes (including the first system process) according to the specification, as described above.

3. DTE Firewall Overview

A DTE firewall runs application-level gateways in controlled DTE domains and also mediates network communications based on DTE security attributes.

Figure 1 shows the general DTE firewall concept of operations. Some hosts behind the firewall are running DTE; there may or may not be DTE hosts outside the perimeter. A

[2] UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Ltd.

[3] Our prototype is based on BSD/OS 2.0 (and recently 2.1), a widely available PC UNIX. Excluding comments, DTE enhancements to the kernel represent approximately 14,000 lines of code, and DTE enhancements to the firewall proxies represent approximately 700 lines of code.

DTE firewall intercepts and mediates all network traffic between internal and external hosts.[4] Based on the DTE policy, the DTE firewall associates DTE domains with non-DTE hosts and associates DTE communication attributes with data received from non-DTE hosts.[5] The DTE firewall and DTE hosts use the communication attributes to ensure that network messages are only received by domains that can appropriately control the network service. If the interior host is a DTE system, it carries the responsibility for confinement of network services: the DTE firewall's role in this case is to coordinate the endpoints' security contexts by passing along the DTE communication attributes. The DTE firewall performs access control on behalf of non-DTE hosts (interior or exterior) by mediating the network messages sent to each non-DTE host based on whether the host's associated domain grants access to the DTE attributes associated with the messages. By coordinating DTE policies between DTE endpoints (and on behalf of non-DTE endpoints), the DTE firewall is positioned both to protect and control exported services and to confine network clients that import services.

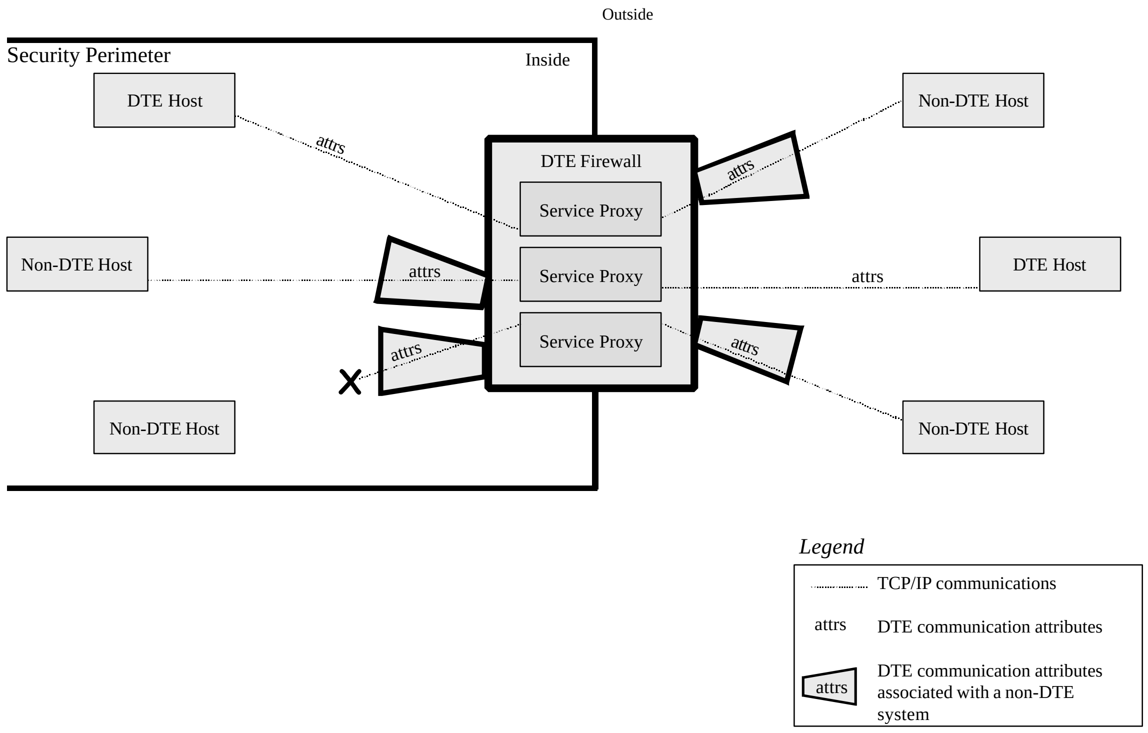

Figure 1: DTE Firewall Concept of Operations

3.1 Controlling Exported Services

Figure 2 illustrates our general strategy for exporting services safely: run network server applications either on DTE firewalls or on DTE hosts behind the security perimeter and use DTE to control access to local resources. In Figure 2, a DTE server system hosting a network server application service communicates through a DTE firewall to respond to service requests from a non-DTE external host Client System. Since the client shown is not

[4] For the work described in this paper, the network traffic is passed "in the clear;" we are currently working on IP-layer cryptographic protection for network communications.

[5] A DTE policy currently associates attributes with packets received from non-DTE systems based on source IP address. While IP addresses can be spoofed in IPv4, the apporach is adequate for prototyping. Stronger mechanisms could be provided using IP-layer cryptography, such as that proposed for IPv6.

running DTE, the firewall associates t and domain sd with messages received from it. The DTE firewall relays the DTE communication attributes to the server: these attributes establish DTE access controls consistent with the level of trust placed in the client by the firewall.

When a client attempts to initiate a connection with the server, the inetd daemon on the firewall runs the netacl program which determines whether communication is allowed between the client and server hosts[6] and executes the proxy application for the specified protocol. The netacl program executes the proxy in the proxy_d domain, which is specific to the protocol (e.g., the HTTP proxy runs in the domain http_d and has DTE access permissions sufficient to pass DTE communication attributes between the client and the server.[7] The proxy_d domain can be configured to control which clients may use the service.

At a high level, the proxy's algorithm is simple:

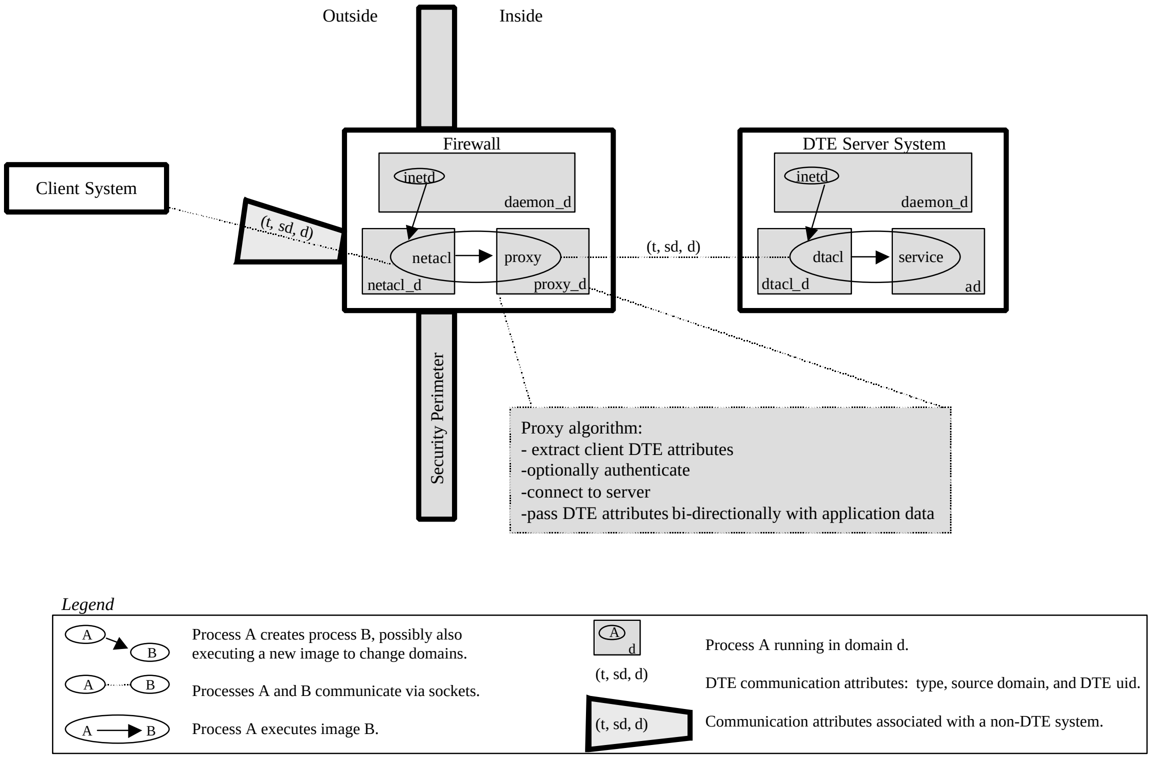
Outside
Inside
Firewall
DTE Server System
Client System
inetd
daemon_d
inetd
daemon_d
(t, sd, d)
netacl
proxy
netacl_d
proxy_d
(t, sd, d)
dtacl
service
dtacl_d
ad
Security Perimeter
Proxy algorithm:
- extract client DTE attributes
-optionally authenticate
-connect to server
-pass DTE attributes bi-directionally with application data
Legend
A
B
Process A creates process B, possibly also executing a new image to change domains.
A
B
Processes A and B communicate via sockets.
A
B
Process A executes image B.
A
d
Process A running in domain d.
(t, sd, d)
DTE communication attributes: type, source domain, and DTE uid.
(t, sd, d)
Communication attributes associated with a non-DTE system.

Figure 2: Exporting Services Safely

1. Extract Client Attributes

The DTE kernel's socket abstraction provides an interface to retrieve the client attributes (t,sd,d) each time data is read from the client socket. Because the attributes are carried in each IP message, they are available for both connection-oriented (e.g., TCP) and connectionless (e.g., UDP) protocols.

2. Optionally Authenticate

[6] This is standard application gateway firewall functionality; the check is based on IP addresses or host names.
[7] This paper assumes a homogeneous DTE policy (having the same domain and type definitions) for all hosts; future work will introduce interactions between hosts funning different policies and dynamic policy reconfiguration.

If the client is not a DTE system, the proxy authenticates the client using the method specified in the firewall's configuration. If the client is a DTE system, the proxy may choose to trust the DTE uid (d) in the communication attributes, which indicates the user's identity as authenticated by the client system. Using a DTE client's authentication attribute increases usability and performance by relieving the client of the need to authenticate for each service initiation (this has resulted in DTE firewall performance increases for some test cases).

3. Connect to Server
The proxy connects to the DTE server system, passing the DTE attributes of the client.

4. Pass DTE Attributes Bidirectionally
The DTE-enhanced proxy scans and copies data as in a conventional firewall except that it also passes the DTE communication attributes through to each endpoint. Also, for each message-receive and message-send operation, the proxy's domain is mediated (by the firewall's DTE kernel) with respect to the type of data being received or sent: the types of data allowed through the firewall therefore can be adjusted by adjusting the types of data that the proxies are allowed to read/write. Because DTE attributes are associated with remote non-DTE hosts by the firewall, the policy also can be adjusted so that some types of data are not sendable (or receivable) from specified hosts or networks. When the external host is running DTE, the control is more fine-grained and communications using specific types of data can be allowed or disallowed for individual domains (or processes) on the remote host.

As shown in Figure 2, when the proxy application connects to the DTE server, the inetd program on the server runs the dtacl program. This program, like netacl, examines the DTE attributes sent in the connect request and then executes the relevant server application service in domain sd, the domain of the client. By running the server application in the client's domain, the system ensures that the client and server run in a compatible security context: if the client is anonymous (e.g., anonymous FTP or HTTP), the server runs in a domain granting very limited access to data on the server system; if the client is a known quantity (perhaps administered by a business partner, or an authorized guest), the server runs in a domain granting access to more types of data.[8] For services that do not restart with each client request, however, a different strategy is needed. These services (e.g., NFS) employ a kernel-supported "auxiliary" domain to further restrict the server when it is working on behalf of a particular client. Because the service must ask the kernel to load and unload the auxiliary domain, this technique is only available to trusted servers.

The strategy of running the server in the client's domain does not necessarily result in least privilege, since the common domain may overstate access needs of the client or the server (i.e., they may not need to share everything). However, the strategy is surprisingly simple and approximates least privilege (in our experience) closely enough to separate user roles and to protect the system. The DTE policy on the server host protects the host from the server application (with root confinement[18]) and also protects the server application from other programs that may run on the server host. Additionally, the types writable by the sd domain indelibly label data that originated from the exterior network. This labeling can be

[8] The access level granted to domains depends upon the DTE policy and is user-configurable.

used to alert users and programs as to the trust to be placed in the data, and also to prevent untrusted imported data from being accidentally executed as a program.

3.2 Controlling Imported Services

A DTE firewall controls imported services using the same mechanisms it uses to control exported services: the DTE firewall relays DTE communications attributes from an internal client system to potential server systems and performs mediation on behalf of non-DTE clients and servers. The overall effect of this mediation is to prevent a client from using a server unless either the server is running in the client's domain or the server runs on a DTE host that is willing to start a copy of the server in the client's domain. As a consequence of these mechanisms, the same client in different domains may have differing levels of access to external services. Users (on a DTE client) therefore may choose restricted domains when accessing untrusted or unknown services (e.g., surfing the web), and choose more privileged domains when accessing important corporate data. Additionally, this constraint ensures, for example, that mail cannot be accidentally sent to competitors and that files cannot be accidentally imported from untrustworthy environments and executed.

In Figure 3, a DTE client system runs a network program that communicates through the DTE firewall to access a service provided by a non-DTE external host. Figure 3 shows the user session running in domain shell_d and the client application running in domain client_d.[9] The domain client_d controls the client application in two important ways: 1) it prevents a successful attack on the client from damaging the client system, and 2) it labels data generated by the client application with a type that identifies the data's origin and indicates the amount of trust that can be placed in it. In addition, the DTE policy on the client system protects the client application from attacks by other running programs. This is particularly useful for preventing software of questionable quality (e.g., the most recent version of a free editor) from accessing the data streams of important client applications such as electronic commerce, banking, or Internet telephone programs.

[9] A DTE client system's DTE policy can be configured to give the user discretion in which domain the client application runs, to require that the client application run in the domain of the user's shell, or to specify automatically a domain in which the client application runs.

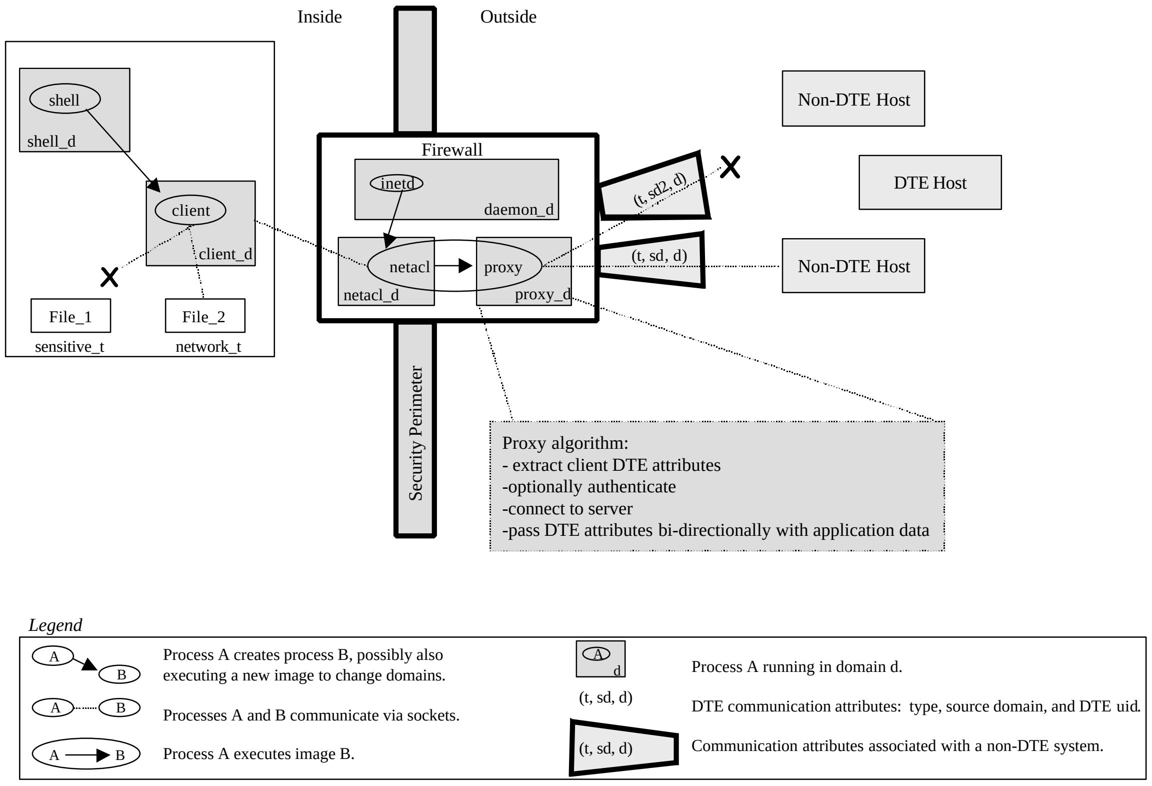
Inside
Outside
shell
shell_d
client
client_d
File_1
File_2
sensitive_t
network_t
Firewall
inetd
daemon_d
netacl
proxy
netacl_d
proxy_d
Security Perimeter
(t, sd2, d)
(t, sd, d)
Non-DTE Host
DTE Host
Non-DTE Host
Proxy algorithm:
- extract client DTE attributes
-optionally authenticate
-connect to server
-pass DTE attributes bi-directionally with application data
Legend
Process A creates process B, possibly also executing a new image to change domains.
Processes A and B communicate via sockets.
Process A executes image B.
Process A running in domain d.
DTE communication attributes: type, source domain, and DTE uid.
Communication attributes associated with a non-DTE system.

Figure 3: Importing Services Safely

As shown in Figure 3, the client application communicates via sockets with the DTE firewall which then communicates with the external server. Since the serving host is not running DTE, the DTE firewall prevents communication with the serving host unless the DTE firewall associates the client's domain with the host.

4. Network Services Evaluation

To assess the overall impact of DTE firewalls and selected hosts, we have informally evaluated the remote login (rlogin), TELNET, mail, FTP, NFS, and HTTP network services running through a DTE firewall based on security, preservation of functionality, compatibility with non-DTE hosts, and performance. For purposes of evaluation, we used the BSD/OS DTE prototype system and the TIS Firewall Toolkit.[10]

4.1 Security

Network attacks often exploit subtle weaknesses in programs that allow attackers to misuse program access rights, gain control over systems, steal or destroy data, and deny access to authorized users. The effectiveness of such attacks can be generically reduced if programs execute with the minimum access rights required to perform their functions. Our informal metric for evaluating the security of a DTE firewall, therefore, is the extent to which it restricts the access rights used by programs running on the firewall or on DTE-aware systems that communicate with the firewall. We have identified three primary areas where program authorizations are reduced by DTE:

[10] This research uses version V1.3 of the TIS Firewall Toolkit.

Confined Proxies: A DTE firewall confines each network proxy in a separate domain. In general, the domains used to confine proxies prevent write access to system and administrative data and also prevent proxies from running other programs. This confinement protects the firewall itself from a possibly subverted proxy: for example, if the FTP proxy is compromised, it cannot alter or view system files. When proxies are trusted to propagate type labels of user data that passes through them, a faulty or subverted proxy may cause user data to be mislabeled, however this mislabeling can be limited: a proxy can only attach a type label if the proxy's domain grants ``w'' access to the label. Consequently, the firewall's DTE policy can be configured to control the discretion afforded a proxy as well as to restrict which types of data are allowed to flow through different network services.

Protected Servers On the Firewall: A DTE firewall can run network services in domains that protect their data and program files from network-based attack. By running services on the firewall (or, for load-sharing purposes, on companion DTE server hosts), the integrity of network services can be substantially improved even though the services are made available to the Internet. While running possibly vulnerable network services on a conventional firewall would pose an unacceptable security hazard, a DTE firewall can safely execute such services because their access rights are limited via DTE.

Defense in Depth: As discussed in section 3.1, DTE firewalls coordinate the features of DTE servers behind a security perimeter to restrict the access rights of programs that are run on behalf of external clients. When DTE servers are available, this strategy ensures that processing carried out on behalf of external clients is controlled in appropriate DTE domains that prevent clients from tricking interior services into granting unauthorized access. This technique can be used to prevent unauthorized export of sensitive data through ubiquitous services, such as email, and to selectively export NFS hierarchies to the Internet. Furthermore, this strategy typically does not rely on the correctness or security sophistication of services exported through the DTE firewall since the control is enforced by the DTE on the serving hosts.[11]

The DTE firewall policy used to conduct our experiments consists of 105 lines of DTEL specification on the firewall and 122 lines of DTEL on supporting DTE hosts. The addition of a new network service or domain adds typically only 5 or 6 lines to these specifications. Additionally, small extensions to domains in the policy can be made that apply uniformly across the specification and expand or constrict access rights for all network services. As a result, DTE policy complexity can be controlled, resulting in enhanced assurance that protections are maintained as new network services are added. The combination of these techniques has significantly reduced the access rights used by software running on (or through) the firewall while allowing the software to function correctly. Based on our informal metric, we belive that these techniques have, in total, significantly increased network enclave security.

4.2 Functionality

For importing services, functionality is rarely affected. In services such as rlogin and TELNET, when the client is a DTE system, user authentication can be supplied automatically by the client DTE system and the proxy can accept and use this authentication instead of requiring additional authentication. Under some circumstances,

[11] The exception is the NFS daemon, which for performance reasons is implemented as a trusted server.

this can increase usability. The DTE uids, however, must be set up in the firewall configuration; this adds a small amount of administrative overhead. Services such as HTTP and FTP can be made more widely available because of the reduced risk of malicious programs (e.g., applets).

Functionality for exported services has increased. With the additional security of running a server in a domain restricted according to trust level of the client, the server no longer needs to be located outside the firewall. Instead it can be on a system behind the firewall, or even on the firewall itself. Furthermore, with the server starting in the domain of the client, it is feasible to grant access to different classes of information based on that domain -- something very useful for HTTP and FTP. In this way, for example, an anonymous FTP server could regulate access to files without resorting to multiple servers or hidden files. Also, a single HTTP server could provide sensitive data to users in trusted domains and general data to users in untrusted domains without fear of having the data compromised or altered. NFS requires slightly more administrative overhead, but becomes safely available - something not possible before.

4.3 Compatibility

Each service can interoperate either with DTE or non-DTE systems; furthermore, the application-level proxies revert to standard Firewall Toolkit behavior when run on a non-DTE kernel. The use of IP options to carry DTE information removes the need for changing any of the protocols. With the exception of the NFS server, which is kernel-resident in UNIX, few of the client or server applications have been changed to function with DTE firewalls. Mail final delivery agents were modified to be cognizant of the different types associated with a user's mailboxes. Also, the rlogin server was modified to take advantage of DTE authentication mechanisms.

Some of the services running under DTE require changes in the administrative configuration. For example, external NFS clients must explicitly name the firewall host as the server whose file systems they wish to mount, since they cannot know the name of the server behind the firewall.

4.4 Performance

To evaluate the performance of DTE and DTE Firewalls, we have constructed a testbed consisting of three Pentium 166MHz machines on an isolated Ethernet running BSD/OS 2.0 and version ``straw_19+'' of the DTE prototype system.[12] We ran each test on a number of configurations where configuration is a triple (clnt, fw, srvr) in which ``y'' indicates a system running DTE and ``n'' indicates a host not running DTE (so (n,y,n) is the configuration where only the firewall is running DTE). The performance of mail was not measured since it is not interactive.

For rlogin, TELNET, and FTP, we used an Expect script to repeatedly perform the following steps. First, invoke the client application, specifying the firewall as the destination. Next, authenticate the user on the firewall.[13] Then, connect to the server, again authenticating the user. Transfer a variable amount of data from the server through

[12] This is the 19th internal version of the BSD/OS-based DTE prototype with some performance enhancements incorporated; many more enhancements could be added in future versions.

[13] When coming from a DTE client, rlogin and TELNET authentication is performed automatically, using the DTE-protected uid.

the firewall to the client, ranging from 0 bytes to 5 MB (except FTP, for which we also tested 32 MB transfers). Finally, log off the server, also terminating the connection with the firewall. Performance numbers were calculated by averaging results from 20 iterations of each test.

For HTTP, we used ZeusBench[14] a standard benchmark. ZeusBench connected to the server via the HTTP gateway, retrieved a specified web page, and closed the connection; the gateway required no authentication. We varied the concurrency, the document length, and the number of requests (1000 requests for 1K documents, and, to save time, 32 requests for 50K documents).

As shown in Figure 4, DTE overheads for rlogin, TELNET, and FTP are modest, with a maximal impact of 13% degradation in the worst case; with additional performance optimization, these could probably be reduced. As is shown in the table, performance actually increases for rlogin and TELNET when the client is running DTE. This is because the DTE client passes a DTE uid which the firewall can accept instead of performing costly authentication. This performance increase does not manifest for FTP because the FTP daemon has its own (always invoked) built-in authentication which we did not disable.

Unlike rlogin, FTP, or TELNET, the HTTP service is approximately 50% slower in the worst case. After analysis, we believe this performance decline for HTTP is somewhat artificial, resulting from the low-performance implementation of the HTTP application gateway in the Firewall Toolkit, which does a separate read() and

Protocol		Data Transferred	Baseline	Percentage Change (%)			
			(n,n,n)	(n,y,n)	(n,y,y)	(y,y,n)	(y,y,y)
rlogin		0K	1.98	0	0	-15	-10
		200K	3.43	6	8	-16	-23
		500K	5.33	<1	<1	-14	-16
		5MB	32.21	<1	<1	-2	-2
TELNET		0K	6.79	<1	<1	-15	-12
		200K	7.79	<1	2	-10	-10
		500K	9.89	1	1	-9	-7
		5MB	41.36	<1	1	-2	-1
FTP		0K	1.98	9	13	6	11
		200K	3.98	4	6	4	5
		500K	4.59	3	4	3	4
		5MB	14.23	3	6	<1	3
		32MB	70.33	2	2	-1	<1
HTTP	Concurrency Level 4	1K	355.81	8	13	12	15
		50K	64.71	71	89	92	94
	Concurrency Level 8	1K	199.20	22	24	26	25
		50K	60.23	75	94	97	114

Figure 4: Raw Performance in Seconds and DTE Overheads

write() system call for each byte transferred. This overstates DTE system call overheads

[14] ZeusBench version 1.0 is copyright Zeus Technology Limited 1996.

because, in this test, the system spends most of its time dispatching system calls that each do very little work. More recent application gateways, such as Gauntlet's[15] perform more efficient I/O; we expect that DTE performance for those gateways should approximate the DTE performance for rlogin, TELNET, & FTP.

For NFS, we used Iozone and NFSstones, two widely-used NFS benchmark packages. The Iozone package tests sequential file I/O by writing a 64 MB sequential file in 8 K chunks, then rewinds it, and reads it back (i.e., it measures the number of bytes per second that a system can read or write to a file). The size of the file was set large enough to prevent the cache from dominating the results. NFSstones creates and deletes many directories, then does a variety of file accesses, including writes, sequential reads, and non-sequential reads. Using these results in a formula, it generates a single numeric indicator of relative NFS performance.

		Baseline	Percentage Change (%)	
		(n,n,n)	(n,y,n)	(n,y,y)
Iozone	Bytes/Second Written	107,372.50	-4	-20
	Bytes/Second Read	409,430.50	-28	-38
NFSstones	NFSstones/Second	69.83	-18	-38

Figure 5: NFS Test Results (Larger Numbers Indicate Better Performance)

As shown by the results in Figure 5, performance of writes under NFS is moderately affected by the addition of DTE to the firewall; adding DTE to the server produces a higher impact on performance, with a 20% performance hit. Reads under NFS, however, dominate NFS performance, with a slowdown of 38% when both the firewall and the server are DTE hosts. However, neither the NFS application-level gateway nor the DTE-enhanced NFS server is optimized. Two possible locations for performance degradation in the NFS server are the double mediation necessary because of the primary/auxiliary domain combination and the manipulation of additional file handles needed for DTE mediation in NFS-mounted files.

5. Related Work

DTE firewalls are related most closely to firewall techniques [6, 5], mandatory access controls [2, 4, 3, 10], and type-enforcing systems [4, 12, 14, 17, 20].

There are three fundamental types of firewalls: packet-filtering, circuit gateway, and application gateway. Packet-filtering firewalls allow packets to pass through only if they satisfy a set of filtering rules based on packet direction, physical interface, and a variety of packet fields (e.g., source address, destination address, UDP/TCP port numbers, etc). Circuit gateway firewalls force all TCP connections to go through an intermediary process which performs initial access control and then copies (and perhaps audits) data streams. Application gateway firewalls force data streams penetrating a firewall to be processed on a per-protocol basis by a trustworthy application proxy program that audits protocol usage and possibly screens damaging data. DTE could be added to either packet-filters or circuit gateways; however, we have incorporated DTE into application gateway firewalls because

[15] Gauntlet is a registered trademark of Trusted Information Systems, Inc.

they provide greater opportunities to explore interactions between DTE and individual protocols.

Mandatory access controls, sometimes called ``rule-based" controls, are centrally configured by system managers and then enforced on ordinary users and their software. A variety of access control techniques [2, 4, 3, 10, 7, 1] provide centralized control over resources. In general, DTE policies are a proper superset of the DoD lattice model [2] and its integrity variation [3]: DTE can be configured to provide a lattice but also can enforce non-hierarchical security policies such as assured pipelines [4] that drive information through policy-specified pathways of arbitrary connectivity and complexity. DTE also can be configured to provide integrity categories as in [10] and to support the transformation procedures and constrained data items of the Clark/Wilson model [7].

A number of systems have implemented type enforcement, including the Secure Ada Target[4] (later renamed LOCK [14]), Trusted XENIX [17], and DTOS [8], which adds type enforcement to Mach port, task, and virtual memory abstractions. Type enforcement also has been integrated into at least one Internet firewall product, the SCC Sidewinder[16] system, [16] an embedded turnkey system employing a fixed, vendor-supplied access control configuration. DTE [16], an enhanced form of type enforcement, was first demonstrated on an OSF/1 UNIX prototype and later ported to BSD/OS.

6. Future Directions

This paper discusses the first phase of our work: a simple, manually-administered, DTE firewall prototype. We plan two additional phases to increase the user-friendliness and flexibility of the prototype.

For the second phase, we have formulated simple DTEL modules for organizing policies into more maintainable segments. We also have begun formulating enhancements that allow DTEL modules to express administrative agreements between network enclaves and to support interactions between compatible but not identical policies. Currently, we are experimenting with loosely-constrained DTEL modules that can be loaded and unloaded while the system runs, at the discretion of the administrator. Since the modules may affect running processes and existing DTE policy elements, the administrator must exercise caution in their use (in this phase). Additionally, we are integrating IP-layer cryptography into the DTE kernel in order to extend DTE protections across unprotected WANs. For this encryption, we chose IPSec in order to provide IPv6 protection mechanisms to IPv4 packets, and are basing our prototype on the Navy Research Labs IPv6/IPsec Software Distribution. As IPv6 becomes more widely available, we will port the DTE network code to use its features.

For the third phase, we have designed an extension to DTEL for "constrained" modules that can be loaded and unloaded without adversely affecting a running DTE policy. In this phase, we plan to develop a trustworthy network service, the Domain Type Authority, which distributes registered DTEL modules to DTE firewalls that would like to set up DTE-controlled shared execution environments. Using the properties of constrained DTEL modules, we plan to extend our prototype to dynamically load and unload constrained modules as network applications establish and break connections with entities external to

[16] Sidewinder is a trademark of Secure Computing Corporation, Inc.

firewall security perimeters. By dynamically loading constrained DTEL modules, we hope to demonstrate flexible automatic loading of security policy components across a network.

7. Conclusions

As currently realized, firewall security perimeters are inexpensive but also inflexible and somewhat weak. A central question for practical commercial security is whether the advantages of firewalls can be preserved while adding additional security controls to protect against inside attacks, to protect sensitive data from export, and to fortify servers against corruption from the network. Adding DTE to firewalls appears to address many security concerns because DTE supports role-based policies that relate resource access to individual responsibilities within the enterprise; and because roles provide an intuitive framework for expressing controlled sharing of resources between organizations. We believe the primary question for enhanced-security firewalls is whether useful security features can be added while preserving the functionality of existing applications and protocols, interoperating with non-enhanced systems and programs, imposing minimal performance overheads, and imposing acceptable administrative costs.

This paper reports our results with a prototype DTE firewall system that runs application gateways in DTE domains, exports network services that run in DTE-protected environments either on the firewall itself or on protected servers, and controls interactions with non-DTE systems. In general, our prototype DTE firewall coordinates DTE security protections of communicating endpoints, ensuring that clients and corresponding servers execute in the same DTE domains. This relatively simple strategy has allowed us to demonstrate controlled sharing through the firewall, protection of sensitive information, confinement of client applications, and increased protection of network servers from client-based attacks. We have found that security can be increased significantly if DTE runs on the firewall and selected server systems. In addition, running DTE on client systems enables support for user roles and confinement of client applications. For rlogin, TELNET, FTP, HTTP, and mail, we have found no significant decrease in functionality or compatibility. The functionality of the NFS protocol has increased since it previously could not be exported safely. For many protocols, performance is not significantly affected by DTE; we believe that significant performance impacts for NFS and HTTP can be eliminated through optimization techniques. Finally, administrative costs, while still an open issue, appear to be manageable since useful DTE policies can be concisely expressed.

References

[1] L. Badger, Daniel F. Sterne, David L. Sherman, and Kenneth M. Walker, "A Domain and Type Enforcement Unix Prototype.", Usenix Computing Systems Volume 9, Cambridge, MA, 1996.

[2] D.E. Bell and L. Lapadula, Secure Comptuer System: Unified Exposition and Multics Interpretation. (Technical Report No. ESD-TR-75-306, Electronics Systems Division, AFSC, Hanscom AF Base, Bedford MA, 1976)

[3] K.J. Biba, "Integrity Considerations for Secure Computer Systems," USAF Electronic Systems Division, Bedford, Mass., ESD-TR-76-372, 1977.

[4] W.E. Roebert and Kain, R.Y., "A Further Note on the Confinement Problem," IEEE 1996 Carnahan Conference on Security Technology, pp 198-203.

[5] D. Brent Chapman, Elizabeth D. Zwicky, "Building Internet Firewalls," O'Reilly and Associates, Inc., 1995.

[6] W.R. Cheswick and S.M. Bellovin, "Firewalls and Internet Security: Repelling the Wily Hacker," Addison-Wesley 1994.

[7] D.D. Clark and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, CA, p. 184, 1987.

[8] T. Fine and S.E. Minear, "Assuring Distributed Trusted Mach," 1993 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, p.206, 1993.

[9] James Gosling and Henry McGilton, "The Java Language Environment: A White Paper," JavaSoft, Mountain View, CA, May 1996.

[10] S.B. Lipner, "Non-Discretionary Controls for Commercial Applications," Proceedings of the 1982 IEEE Symposium on Security and Privacy, Oakland, CA, p.2, 1982.

[11] National Computer Security Center, "A Guide To Understanding Discretionary Access Control In Trusted Systems," NCSC-TG-003, Sept. 30, 1987.

[12] R. O'Brien and C. Rogers. "Developing Applications on LOCK", In Proc. 14th National Computer Security Conferences, pages 147-156, Washington, DC October 1991.

[13] The Object Management Group, Inc., "The Common Object Request Broker Architecture and Specification," Revision 2.0, Framingham, MA, July 1995.

[14] O.S. Saydjari, J.M. Beckman, and J.R. Leaman, "LOCK Trek: Navigating Uncharted Space," Proceedings of the 1989 IEEE Smposium on Security and Privacy, Oakland, CA, p. 167, 1989.

[15] D.L. Sherman, D.F. Sterne, L. Badger, S. Murphy "Controlling Network Communication with Domain and Type Enforcement," TIS Technical Report TISR 523D.

[16] D.J. Thomsen, "Sidewinder: Combining Type Enforcement and UNIX," Proc. 11th Computer Security Applications Conference, Orlando, FL, December 1995.

[17] D. Sterne. "A TCB Subset for Integrity and Role-Based Access Control". In Proc. 15th National Computer Security Conference, pages 680-696, Baltimore, MD, 1992.

[18] K. Walker, Daniel F. Sterne, M. Lee Badger, Michael J. Petkac, David L. Sherman, Karen A. Oostendorp, "Confining Root Programs with Domain and Type Enforcement.", Proceedings of the 6th Usenix Security Symposium, San Jose, CA, 1996.

[19] Joseph Williams, "Bots and Other Internet Beasties," Sams.net Publishing, 1996.
[20] S. Wiseman, "A Secure Capability Computer Systems," Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, CA, p.86, 1986.

This research was supported by ARPA contract DABT63-95-C-0018.
Republished here with permission from IEEE Computer Society.

A Scalable Approach to Access Control in Distributed Object Systems

by Gregg Tally, Daniel Sterne, Durward McDonell, David Sherman, David Sames, Pierre Pasturel, and John Sebes

1. Introduction

Although distributed object technology has matured greatly during recent years, the lack of practical, integrated security mechanisms remains an obstacle to its use in many application domains. CORBA[17] and Java Remote Method Invocation (RMI)[18] are two of the more popular distributed object technologies. Version 1.0 of the CORBA Security (CORBASec) specification was initially released in January 1996. However, commercial product releases that fully comply with its Level 2 (general-purpose) requirements are just beginning to emerge. Moreover, even Level 2-compliant products may prove inadequate because Level 2 requirements in some functional areas are not sufficiently stringent. In particular, CORBA Security does not establish requirements for security management mechanisms. These mechanisms can have a major impact on the usability and scalability of the implementation. Java security, as represented by the JavaSoft's JDK1.2 release candidates, is still undergoing rapid evolution. For example, no facilities are provided for controlling RMI-based access by Java clients to remote Java servers.

Providing practical access control mechanisms for distributed objects - CORBA or Java RMI - is a particular challenge because the necessary characteristics for such mechanisms are not well understood. One particular problem is that the number of objects and methods in a large Object-Oriented (OO) system may be enormous. Consequently, attempting to associate separate access control attributes with every method of every object causes a proliferation of access control information that can easily overwhelm administrative personnel, resulting in a loss of understandability and manageability. This problem is worsened when access control lists (ACL) are used as attributes [OSG] [DENG]; each ACL can contain many entries (including those that appear contradictory) whose combined effect can only be understood by applying complex precedence and ordering rules [BALD] [DOWN].

This paper describes the results of a DARPA-funded research project to develop access control technology for distributed objects. By access control technology, we mean mechanisms for 1) specifying the sets of objects and methods that automated processes may access, and 2) protecting objects and methods from unauthorized access. The technology we have developed is called Object Oriented Domain and Type Enforcement (OO-DTE). OO-DTE is an outgrowth of earlier research into access controls for secure operating system kernels. [BADG].

OO-DTE was developed with the following goals:

- Object-oriented – It should support OO abstractions and take advantage of OO relationships, especially interface inheritance.
- Scalability and manageability – It must be suitable for controlling access in large distributed applications systems comprising thousands of processes, objects, and methods.
- Fine-grained control – It must allow access to be controlled at the level of individual objects and individual methods.
- Role-based access control (RBAC) – It should support access rules organized according to user roles (job titles or functional responsibilities) rather than user identities or security clearances.

[17] Common Object Request Broker Architecture from the Object Management Group
[18] Java is a product of Sun Microsystems

- Compatibility with commercial products – It should be designed as a plug-in module that can be linked into commercial object request brokers (ORB) and distributed object infrastructure components.
- Transparency – It should support *security unaware* applications, i.e., applications that do not explicitly invoke access control or other security services.

This paper is organized as follows: Section 2 provides an overview of our approach. Section 3 describes DTEL++, the compilable language used in OO-DTE to specify access control policies. Section 4 describes the features and status of our OO-DTE prototypes. Section 5 compares OO-DTE to CORBA Security. Observations on our experience to date are discussed in Section 6. Section 7 provides a summary and discussion of future work.

2. Approach

We begin the overview of our approach with a discussion of background information.

2.1 Background

OO-DTE is an outgrowth of our earlier secure operating system research in which we developed Domain and Type Enforcement (DTE), a set of access control mechanisms for UNIX kernels [BADG][SHER]. DTE was, in turn, based on Type Enforcement, as proposed originally by Boebert and Kain [BOEB]. On a DTE operating system, a security attribute called a type is associated with every file, directory, device, and IP packet. Types are assigned to these system resources according to a site-specific policy. Types are typically assigned to indicate the kind of information contained, its sensitivity, integrity, or origin. A security attribute called a domain is associated with every process. Domains are assigned to processes to indicate the kind of computing tasks they are intended to perform and to constrain their ability to access system resources. Each domain is defined as a collection of access rights. Each right is expressed as the ability to access information of a specified type in one or more access modes, e.g., read, write, execute, send, receive, etc.

One of the innovations of DTE was the use of a compilable high-level language to declare types and domains. This language is called the DTE Language or DTEL. DTEL was also used to assign types to the file hierarchy via a set of general rules with exceptions. This allowed an entire file system, potentially consisting of millions of files, to be "labeled" in a concise and understandable manner by means of a few DTEL statements. This contrasts with ordinary UNIX systems that require an administrator to examine individually the permission bits on vast numbers of files and directories in order to infer how files of different sensitivities are arranged and distributed throughout the file hierarchy.

OO-DTE is an attempt to extend DTE notions to distributed object-oriented systems. We have concentrated initially on developing OO-DTE for CORBA-based systems but are now adapting OO-DTE for Java RMI. OO-DTE includes an extended policy language called DTEL++ that provides constructs for assigning types to methods. DTEL++ also includes rules for propagating default type assignments from modules and interfaces to enclosed methods, and for propagating type assignments to inherited methods. DTEL++ defines two new access modes for methods: invoke (needed by clients) and implement (needed by servers).

The first OO-DTE prototype ran on a DTE kernel and exploited the DTE kernel's access control facilities for interprocess communications [SHER]. This prototype, sometimes referred to as Kernel Level OO-DTE, involved using and modifying the ILU ORB[19]. These modifications were needed to make ILU use the DTE kernel's extended send and receive system calls, which transmit type attributes with messages and prevent senders and receivers from accessing unauthorized information types.

[19] Inter Language Unification (ILU) was developed by Xerox Corporation, Palo Alto Research Center.

Although the DTE kernel provides important security advantages, there is little interest outside the research community in non-standard kernels. To broaden the potential audience for our research we subsequently developed a second prototype called Above Kernel OO-DTE. Above Kernel OO-DTE was specifically designed to run on mainstream commercial operating systems and is the primary focus of this paper.

2.2 Domains, Roles, and Role Authorization

OO-DTE has been designed to support role-based security policies, that is policies that grant users access rights according to their assigned roles (duties) within an organization [NICO]. In our earlier DTE kernel operating systems, each role was represented as a collection of domains. This allowed each task initiated by a user to run as a separate process in a "small domain" providing only the minimum set of access rights needed to accomplish that task. This approach was developed in accordance with the principle of least privilege [SALT]. OO-DTE, however, is designed to run on mainstream operating systems whose kernels lack comparable process spawning and access control facilities. Consequently, in OO-DTE, each role is represented by a single domain; hence, the terms role and domain have become nearly synonymous. The primary difference is that only user processes are associated with roles whereas both daemons and user processes are associated with domains.

In OO-DTE, a user may be authorized for multiple roles, but each process associated with that user is bound to a single role (domain). OO-DTE roles can be constructed hierarchically by defining domains in terms of other domains, in the manner of Baldwin's Named Protection Domains [BALD]. OO-DTE does not provide facilities for dynamic or static separation of duty; unlike some practitioners [KUHN], we regard these as distinct from the fundamental aspects of role-based access control.

Roles, domains, and types provide several levels of indirection in the authorization policy. Users hold certificates that contain a domain as a privilege attribute. The policy defines domains by their access to types. The policy also assigns types to methods. Through these levels of indirection, a user's ability to invoke a method can be determined. Figure 1 shows the relationships between users, domains, types, and methods:

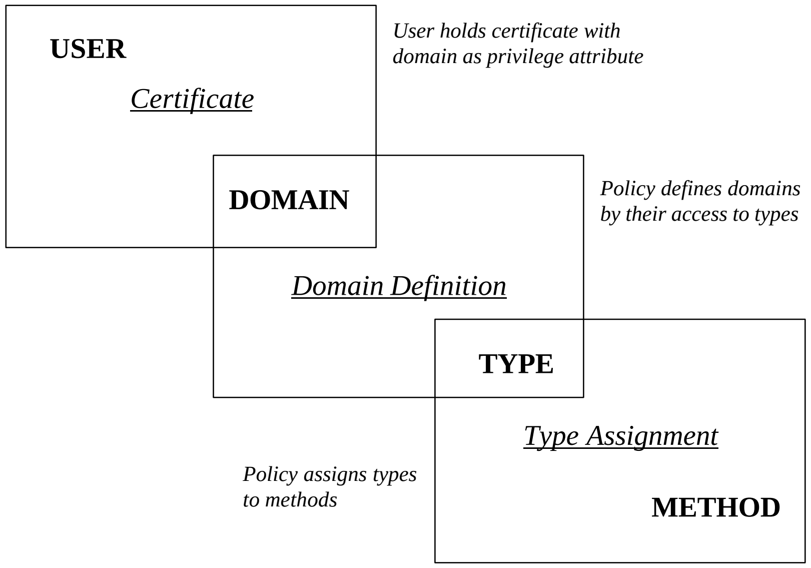
USER
Certificate
User holds certificate with domain as privilege attribute
DOMAIN
Policy defines domains by their access to types
Domain Definition
TYPE
Type Assignment
Policy assigns types to methods
METHOD

Figure 1 - Users, Roles, and Authorization

2.3 Compilable Authorization Language – DTEL++

As with many authorization policies, OO-DTE has requirements to provide both scalability and fine-grained access control. OO-DTE provides a policy language, DTEL++, to meet these requirements. DTEL++ provides a concise format for specifying authorization policy across a large number of methods and users. This is achieved through the use of default module and interface types, recursive type assignments for inherited methods, and role-based access control.

DTEL++ also provides fine-grained access control constructs that can scale to large numbers of objects. Per-object access control is implemented (in part) by aggregating similar objects in "sub-directories" of the object namespace. The authorization policy is uniform for the objects within the sub-directory. New objects named under existing portions of the namespace automatically receive the special access policy.

Some authorization policy mechanisms can achieve fine-grained control, but fail to provide easy inspection of the policy. For example, DCE[20] uses access control lists to express authorization policy [OSG]. File systems often use permission bits, owners, and groups to describe the permitted modes of access by authorized users. In these systems, it is difficult to answer basic questions about the policy, such as "which files is user X permitted to read". Policy languages can avoid this problem. In DTEL++, the policy is initially expressed at a high level of abstraction.

The compiler then derives the policy for specific methods and objects from this higher level abstraction. This is much more suited for audit and inspection than the reverse approach of specifying very fine-grained policy and then attempting to determine if that meets the overall security policy of the organization.

The use of a policy language does not preclude the possibility for using a graphical user interface (GUI) tool to administer the policy. The decision to specify authorization policy through a text-based language or through an interactive software tool is similar to selecting between text-based command shells and graphical user interfaces for operating systems. Many users find that they need both modes of interaction to be productive. DTEL++ provides the power of a policy language approach without prohibiting the development of software tools that can simplify some policy administration tasks.

2.4 DTEL++ Compiler

While policy languages are more flexible than GUI-based approaches, they can be more error prone. For any OO authorization policy, there must be consistency between the policy and the interface definitions. This is achieved in OO-DTE through the DTEL++ compiler. The compiler requires both a DTEL++ policy and the associated IDL files as input. The compiler processes both forms of input and compares them for consistency. Figure 2 shows the inputs to the compiler and the resulting output.

The compiler ensures that methods shown in the DTEL++ policy appear in the IDL. It generates output (in the DTE metadata files) for all CORBA methods in the IDL, as well as standard DTEL policy files that contain the domain definitions. The DTE metadata files are input to the ORB and are not intended for human consumption.

It is not required that there be an explicit rule in the DTEL++ policy for every CORBA method. The DTEL++ language allows type assignments to be assigned recursively from base interfaces to derived interfaces.

Although DTEL++ type assignments are similar in structure to IDL, some characteristics are slightly different. For example, inheritance relationships between interfaces are not stated in

[20] Distributed Computing Environment from The Open Group

DTEL++. For this reason, a 'viewer' tool may be developed to show the DTEL++ policy in combination with the IDL.

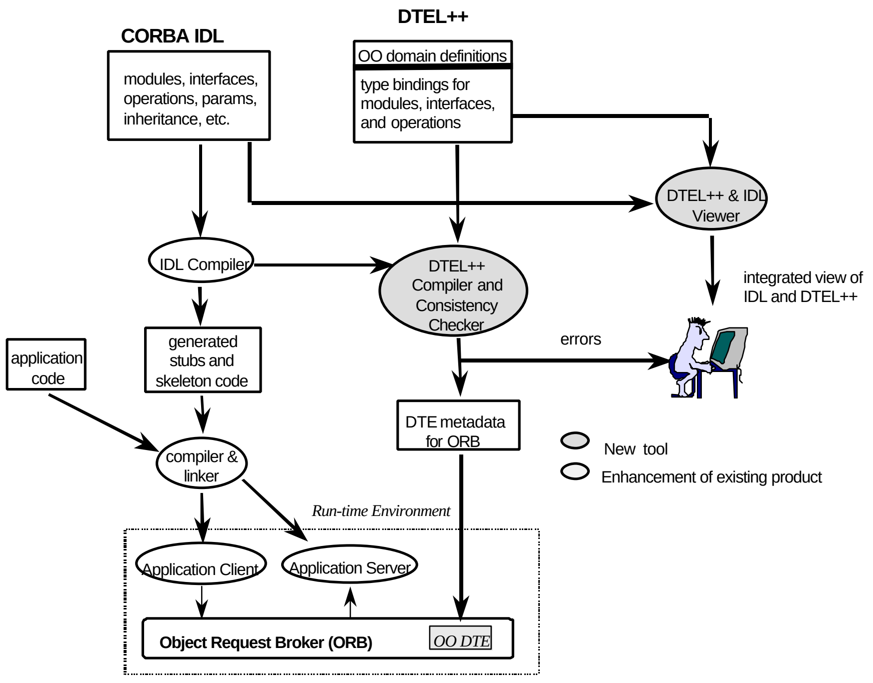
CORBA IDL
modules, interfaces, operations, params, inheritance, etc.
DTEL++
OO domain definitions
type bindings for modules, interfaces, and operations
DTEL++ & IDL Viewer
IDL Compiler
DTEL++ Compiler and Consistency Checker
integrated view of IDL and DTEL++
errors
application code
generated stubs and skeleton code
DTE metadata for ORB
compiler & linker
New tool
Enhancement of existing product
Run-time Environment
Application Client
Application Server
Object Request Broker (ORB)
OO DTE

Figure 2 – DTEL++ Processing

2.5 *Authentication and Authorization Module*

A 'plug-in' to the ORB is responsible for performing authentication and authorization according to the DTEL++ policy. The plug-in is loaded into both the client and server and acts as an extension of the ORB message processing. The plug-in is responsible for:

- mutual authentication between clients and servers, and
- authorization checks on CORBA requests according to the OO-DTE policy.

Authentication is performed when the client attempts to establish a connection with the server. SSL[21] provides authentication and a secure transport. During connection establishment, the client and server exchange X.509 certificates that indicate the domain of each participant. The domain of each participant in the connection is fixed for the duration of the connection.

OO-DTE uses a model of mutual suspicion between the client and server. Authorization checks are performed in the client and server on every method invocation. Clients verify that the server domain is permitted to implement the type assigned to the requested method. Servers verify that the client domain is permitted to invoke the type assigned to the method. This prevents spoofing by malicious

[21] Secure Sockets Layer from Netscape Communications Corporation

processes imitating servers, and provides the traditional protection of server resources from unauthorized clients.

2.6 Policy Distribution

Distributed systems require consistency of security policy across the hosts in the system in order to operate in the intended manner. OO-DTE implements a decentralized approach to access control; access control decisions are made locally to each application process. The ORB plug-in (authorization module) in each application must read the metadata files during initialization. Policy metadata files are stored on each client and server host so that the authorization module has reliable and efficient access to the policy. In order to provide a consistent policy across the distributed system, all policy changes must be propagated to each affected OO-DTE host. Automatic and secure distribution and synchronization of policy updates is required when dealing with large numbers of hosts and metadata files.

3. DTEL++ Policy Language

DTEL++ is the policy language used to specify OO-DTE security policies. It is used to define types, to assign a type to each method that might be invoked, to define domains, and to declare which types each domain can invoke or implement. We will motivate the discussion of the policy language by illustrating how it can be used to control a simple application system.

3.1 Sample Application

Suppose we are writing an application to manage the books in a library. Our system has librarians, patrons, and books. One can enter, remove, and find books in the card catalog, reserve books, and check books in and out. The IDL for this application might look like:

```
   module Library {
        struct BookDescription {...};
        interface Patron {...};
        interface PatronDatabase {...};

              interface Book {
            readonly attribute BookDescription desc;
            void checkOut (in Patron patron);
            void checkIn ();
            long numberAvailable();
            long numberReservations();
            void reserve (in Patron patron);
           };
interface BookDatabase {
            Book newBook (in BookDescription desc, in long copies);
            void removeBook (in Book book);
            Book findByTitle   (in string title);
            Book findByAuthor  (in string author);
            Book findBySubject (in string subject);
        };
    };
```

In our policy, we want to allow both librarians and patrons to find books in the card catalog and to reserve books. Additionally, we want to allow only librarians to check books in and out, to enter new books into the catalog, and to remove books from the catalog.

3.2 Type Definitions and Simple Type Assignments

Every DTEL++ policy must define types. These types are then used to label interface methods so that access to the methods can be controlled. In our policy we define two types, safe_t and restricted_t[22] with the OO_type statement:

```
OO_type   safe_t, restricted_t;
```

Methods that patrons and librarians are allowed to invoke will receive the safe_t type, and methods that only librarians are allowed to invoke will receive restricted_t.

Each method in each interface must be assigned a type. This can happen in one of several ways. The most straightforward way is with an explicit assign statement for the method. Since patrons can safely be allowed to lookup books in the catalog, the type safe_t is assigned to the method findByTitle() by the assign statement:

```
assign   safe_t   findByTitle;
```

A second way of assigning types to methods is to assign a default type to methods, as in:

```
assign   restricted_t _DEFAULT;
```

By declaring a default type of restricted_t any method which does not receive a type via an explicit assign statement, as shown previously, will receive the type restricted_t. This allows us to construct our policy so that patrons can perform only those methods that have been explicitly permitted to them.

Other ways of implicitly assigning types will be discussed in a later section.

3.3 Domain Definitions

A DTEL++ policy must define domains and declare the permissions of each domain. For each type defined in the policy, a given domain can have invoke permission, implement permission, neither permission, or both permissions. Only those permissions explicitly stated in the domain definition are granted to the domain. In our example application, we can allow patrons to perform the safe operations, and no others, with the domain declaration:

```
domain patron_d = (invoke->safe_t);
```

We permit librarians to perform both safe and restricted operations with:

```
domain librarian_d = (invoke->safe_t, restricted_t);
```

Domains can be concatenated. Since the librarian_d domain has a superset of the privileges of the patron_d domain, it could have been defined as:

```
domain librarian_d  =  patron_d, (invoke-
>restricted_t);
```

While the impact upon our example is minimal, had the definition of the patron_d domain been substantially more complex, the above construction of the librarian_d domain would still allow the policy to show very clearly the relationship of the two domains. This allows one to express larger policies in a more concise, and more understandable way.

For an example of both invoke and implement permission, consider augmenting the server_d domain. We may want to give it the invoke privilege for the safe_t type if it has to invoke some of the listed methods internally to do its job. We would change the domain statement to this:

[22] We have adopted the convention that the suffixes "_t "and "_d" denote type and domain identifiers respectively.

```
          domain server_d =   (invoke->safe_t),
                    (implement->safe_t, restricted_t);
```
The software that maintains the library database must implement all of the methods and requires the following domain definition:

```
          domain server_d = (implement->safe_t, restricted_t);
```

3.3 Sample Policy

Here is the DTEL++ policy for our library system:

```
          OO_type safe_t, restricted_t;
          module Library {
              assign restricted_t _DEFAULT;
       interface Book {
           assign safe_t { _get_desc, numberAvailable,
                               numberReservations, reserve };
       };

              interface BookDatabase {
                  assign safe_t { findByTitle, findByAuthor,
          findBySubject };
       };
   };

     domain patron_d    = (invoke->safe_t);
     domain librarian_d = (invoke->safe_t, restricted_t);
     domain server_d    = (implement->safe_t, restricted_t);
```

The first line declares the two types that we are using. Next, we open the Library module and declare that restricted_t is the default type for all methods in the module. Then we open the Book and BookDatabase interfaces and explicitly assign the safe_t type to methods that may be invoked by patrons. Finally, we declare our
domains: applications run by patrons will run in the patron_d domain and be allowed to invoke only safe_t methods, librarian applications will run in the librarian_d domain and be allowed to invoke safe_t and restricted_t methods, and any servers will run in the server_d domain and be allowed to implement both safe_t and restricted_t methods.

When an assign statement for a default type appears within the scope of an interface statement, the default type applies to the methods of that interface. When such an assign statement appears outside the scope of an interface statement but within the scope of the module statement, the default type is accepted as the default type by all interfaces within the module that do not contain their own default assign statement to override it.

In our example, we declared a default type of restricted_t for the entire Library module. All interfaces within the module will have restricted_t as their default type since none of our interfaces contain an assign statement to reset the default type within the interface. All methods in the Patron and PatronDatabase interfaces, which are not even mentioned in the policy, will receive the restricted_t type by default. In the Book interface we assigned the type safe_t to some of the methods, but since we did not explicitly assign types for the checkIn() and checkOut() methods, they both receive the interface default of restricted_t.

3.4.1 Inheritance

By default, DTEL++ method types are assigned recursively from base interfaces to derived interfaces. For example, assume an interface ChildrensBook that derives from the Book interface described above. The ChildrensBook::checkOut() method would automatically receive the same type, restricted_t, that was assigned to the checkOut() method in the Book interface. It is also possible to explicitly assign a type to the ChildrensBook::checkOut() method so that the inherited type is overridden.

The inheritance of type assignments in DTEL++ can be fine-tuned through the use of the –i, -l, and –f flags. These flags appear immediately after the keyword assign and modify the effect of the type assignments on inherited methods or derived interfaces.

3.4.2 Per Object Access Control

DTEL++ allows you to have different access policies for different objects with the same interface, if you give names to these objects. Objects with different names can then be treated differently.

For example, there may be some books in a library that no one may check out, perhaps because they are antiques. Instead of having to define a new AntiqueBook interface, we can leave the IDL for the application unchanged and add the following lines to our DTEL++ policy:

```
OO_type null_t;   // No one has permission to invoke “null_t”.
...
module Library {
 ...
    template AntiqueBook:
    interface Book {
        assign null_t checkOut;
    };

            assign AntiqueBook /Books/Antique/;
        ...
 };
```

The collection of type assignments for a given interface is called its *default type template*. This includes both explicit and implicit type assignments received through inheritance relationships or default types. An object that is not named has the *default type template* applied to it. The example above defines a *named type template* AntiqueBook. The *named type template* applies only to objects with names that fall under the namespace assigned to the type template (/Books/Antique/).

Object names are arranged in a directory-like structure, and are given to the objects by security aware object factories. In our example, an antique book should get a name such as “/Books/Antique/1003” (for book #1003 in the catalog, for example), while a regular book should get a name such as “/Books/1351” (or possibly no name at all). The DTEL++ line

```
        assign AntiqueBook /Books/Antique/;
```

says that any object with a name that starts with “/Books/Antique/” should have its methods typed as in the AntiqueBook template, rather than the standard (unnamed) Book template.

4. Above Kernel OO-DTE Prototype

Two prototype implementations of OO-DTE have been built and demonstrated. The first version was implemented using the ILU ORB on a DTE-enhanced version of BSD/OS.[23] This version is called Kernel Level OO-DTE. The second version, which is the subject of this section, was implemented using Orbix (Iona Technologies) on Solaris (Sun). It is called Above Kernel OO-DTE.

4.1 Design Overview

Above Kernel OO-DTE has been implemented using single-threaded Orbix for C++ on Solaris, with the add-on SSL package included in Orbix OTM.[24] Commercial ORBs, like Orbix, provide a "plug in" interface for extending ORB functionality. The CORBA specifications call these plug-ins interceptors. In practice, ORB vendors have implemented their own proprietary interfaces for providing this functionality. Orbix provides two proprietary implementations of interceptor-like functionality:

- *Transformers* manipulate raw communications buffers in close proximity to the send and receive events. This is useful for performing encryption and decryption, and in particular, can be used to implement SSL.
- *Filters* inspect and modify requests and replies at an object level. CORBA standards define a Request object as part of the Dynamic Invocation Interface (DII). This object provides access to the components of a request, such as the target object reference, arguments, and the method name. OO-DTE relies on *per-process filters* to perform access mediation. Per-process filters are invoked on every invocation request and reply message.

In OO-DTE, the authentication mechanism is responsible for authenticating the privileges (i.e., domain) asserted by each of the communicating peers. SSL was selected as the authentication mechanism for OO-DTE, primarily because it is the most widely supported security technology for CORBA. It also provides the best hope for near-term interoperability between ORBs.

Figure 3 shows the relationships among the Above Kernel OO-DTE components:

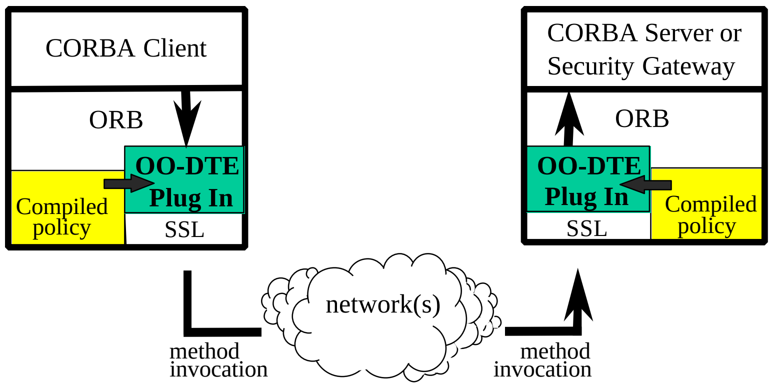

Figure 3 - OO-DTE Components

OO-DTE implements a secure invocation by performing the following steps:

1. <u>Authentication</u> – When the client makes an initial request on a server, the client and server exchange domain information via SSL and setup a connection.

[23] BSD/OS is a product of Berkeley Software Design, Inc.
[24] Orbix Transaction Monitor is a product of Iona Technologies

2. Authorization of the server by the client – Performed by the client-side ORB plug-in.
3. Transmission of request via secure transport – Using previously established SSL connection.
4. Authorization of the client by the server – Performed by the server-side ORB plug-in.
5. Implementation of the request – Performed by the server application code.
6. Return of the reply to the client – Performed by the ORBs; not mediated by the ORB plug-in.

The OO-DTE filters examine each method request to determine whether it is permitted according to the DTEL++ security policy. For each request, the server filter checks that the client is authorized to invoke the method type. The client filter checks that the server is authorized to implement the method type.

4.2 Authentication

SSL provides the authentication mechanism for Above Kernel OO-DTE. X.509 certificates are issued to principals and servers. The certificates contain a privilege attribute for a single domain.[25] An OO-DTE process is currently limited to having a single certificate, which implies that it has access to only one domain. However, a user authorized to act in multiple roles can concurrently run processes in different domains.

SSL handshaking is performed whenever a client and server establish a connection. The handshaking process allows certificates to be exchanged. Each peer retains the domain information of the other. The domain information is associated with the connection for the lifetime of the connection.

The server authorization module uses the client domain to control access to methods. The server domain information is also important as protection against malicious applications impersonating servers. Clients can validate the identity of the server, as well as ensure that the server is authorized to perform the methods that it advertises.

4.3 Authorization

Authorization is performed in the OO-DTE plug-in (Orbix filter) in both the client and server. The plug-in performs the following:
1. Determines if object is named, and extracts name if necessary.
2. Determines method type based on method, interface, and optional object name.
3. Performs access decision based on peer domain, mode, and method type.

5. Object Naming

Per-object access control relies on assigning unique identifiers (object names) to objects and organizing the object names in a hierarchical order. CORBA deliberately avoids providing unique identifiers to objects, so OO-DTE was required to invent a specialized naming mechanism. There are several requirements for this mechanism:

- An object can have at most one name.
- There must be a means for the plug-in to determine the name of an object given a reference to the object, preferably without making a remote invocation.
- The names must be organized in a hierarchical manner to allow easy association with type templates.
- Object names must be assigned at object creation and thereafter be immutable. Objects cannot be named after creation.

[25] In the initial prototype of Above Kernel OO-DTE, the Organizational Unit of the Distinguished Name contains the domain information. Future implementations may use attribute certificates or extension fields to avoid conflicts with other users of the Organizational Unit field.

An obvious candidate for naming objects is the CORBA Naming Service. However, the Naming Service does not provide a two-way association between objects and names. Given a name, it is possible to find the associated object using a Name Server. However, it is not possible to determine an object's name from a reference to the object. Furthermore, CORBA Naming provides for a many-to-one mapping between names and objects.

The Orbix OO-DTE prototype implementation uses a combination of mechanisms to assign names to objects. First, the object name is bound to the object in a secure implementation of the Naming Service. The Naming Service provides structure to the naming hierarchy and ensures that object names are not re-used. Second, a 'stringified' form of the object name is stored in the *marker* portion of the object reference to provide the two-way association between the name and object. This allows the object name to be extracted from the object reference by a plug-in on the client side. Finally, the object name is also stored as a property of the object using the CORBA *Property Service*[26]. Object factories are responsible for creating and naming the object with all of the correct mechanisms. Per-object access control requires a security-aware application. However, the impact is limited to the factory.

5.1 Type Determination

A component called the Type Deriver uses the DTE metadata output from the DTEL++ compiler to calculate the type associated with a method. If the metadata does not contain a match for a given method, the type is undefined. The ORB is not permitted to continue with an invocation request if the type is undefined. The Type Deriver returns a permission exception to the ORB.

The Type Deriver can determine method types on a per-object basis. DTEL++ policy allows object names to be assigned to type templates. The object name is extracted from the object reference and passed to the Type Deriver. The Type Deriver uses the object name to find the assigned type template. Once the type template for an object is determined, the rest of the type calculation process is the same as with unnamed objects.

6. Access Control Decision

In the client-side filter, the authorization module determines if the server's domain has implement privileges for the request type. In the server-side filter, the authorization module determines if the client's domain has invoke privileges for the request type. If either check fails, a standard CORBA:NO_PERMISSION exception is returned to the client. Access control is not performed when returning reply messages.

6.1 Policy Distribution and Synchronization

OO-DTE takes a decentralized approach to authorization decisions. Therefore, policy changes, in general, must be propagated to multiple hosts to ensure consistency in policy enforcement. The prototype implementation provides a central point of administration and control with limited fault tolerance capability. Policy update notifications are pushed from the central policy administrator to the individual OO-DTE hosts and applications, providing loose synchronization among OO-DTE hosts within an enclave.

[26] It is probably not necessary to use the Property Service for Orbix. Although Orbix provides a proprietary interface for changing an object's marker, effectively only a server can do it. It would be advantageous to remove the requirement for using the Property Service to avoid having to impose requirements on application IDL.

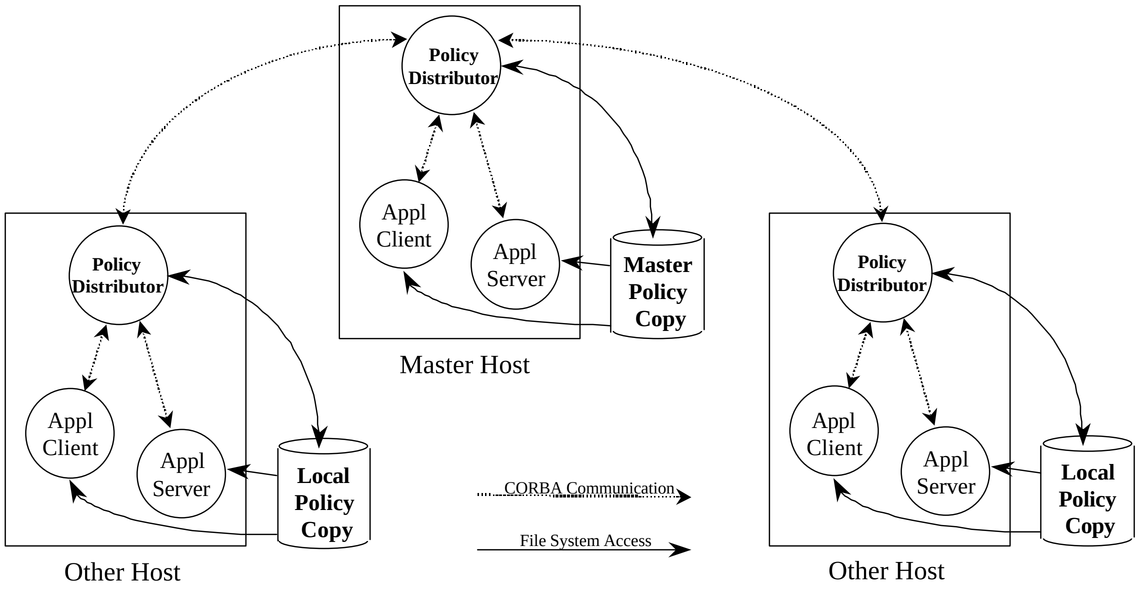

Figure 4 - Policy Distribution Overview

There are four major components for policy distribution within OO-DTE:

- Policy Directories - A policy directory resides on each host in the enclave. A configuration file in the directory identifies one host as the master. The master contains the master copy of the policy that governs the enclave. The other hosts contain local copies of the policy retrieved from the master.
- Administration Program - The Administration Program runs as a CORBA application on the master host. It is used to communicate policy changes to the Master Policy Distributor.
- Policy Distributor Processes – A Policy Distributor runs on each host in the enclave. The one running on the master host is the Master Policy Distributor, and notifies the Local Policy Distributors when the policy changes. Each Policy Distributor copies the latest policy and configuration files from the Master Policy Distributor, keeping its Policy Directory up to date. All Policy Distributors inform their local CORBA applications when the policy changes.
- Policy Distribution Library Routines - Policy distribution library routines are linked into each CORBA application. At application startup these routines register with the Local Policy Distributor to accept notifications when the policy changes, so that the distribution routines can re-read the policy.

This design for policy distribution provides the following features:

- Any type of policy can be distributed – The policy distribution mechanism is separate from the policy enforcement mechanism. A configuration file specifies the files and directories that comprise the policy.
- Policy distributors need not be available to run applications - Each host maintains a local copy of the policy. OO-DTE can be configured to permit applications to run without the policy distributors in the event that either the local or master policy distributor is unavailable.
- Synchronization of policy versions - At startup, Local Policy Distributors always check that the host has the current version of the policy.
- Automatic notification and distribution of policy updates – At startup, Local Policy Distributors register with the Master Policy Distributor to receive notifications of policy updates. Similarly, applications register with their Local Policy Distributors to receive policy update notifications.
- Policy distribution is via secure communications – The CORBA OO-DTE mechanisms are used to control policy distribution between hosts and to control communication with applications. SSL provides authentication and integrity to control policy update notifications and policy distribution.

6.2 ORB Gateway

OO-DTE concepts have also been implemented at the enclave boundary level through the *ORB Gateway*.The ORB Gateway is positioned as an enclave boundary control device.The prototype configurations use the enclave firewall to redirect IIOP requests to the ORB Gateway, rather than the destination server. The ORB Gateway performs its access control functions prior to forwarding the request to the server. This configuration does not require modifications to the object reference or IIOP messages.

The prototype implementation of the ORB Gateway supports several security technologies: SSL, DCE, and authenticated IP. In addition, it supports unauthenticated IP. When the ORB Gateway receives a CORBA request, it derives a domain from a combination of the credentials associated with the request and the trust rating associated with those credentials. Once a domain is established, the rest of the access decision process proceeds as described in Above Kernel OO-DTE.

7. Comparison to Access Control in CORBA Security

In this section we briefly relate OO-DTE to the access control terminology and features described in the CORBA Security Specification (CORBASec) [CORB].

7.1 Terminology

According to CORBASec, principals (e.g., processes acting on behalf of users) possess *identity attributes* and *privilege attributes*. For example, group membership may constitute a privilege attribute. A CORBASec *access policy* translates privilege attributes into *rights*, which are loosely analogous to access modes. Rights belong to *rights families*. The specification defines a standard rights family that includes "get", "set", and "manage" rights. Additional rights families may be defined, but doing so is discouraged in the specification. In order to invoke a method, a principal must possess the *Required Rights* for that method; these depend on the *Security Policy Domain*[27] associated with the target object. Required Rights specify the rights that a principal must possess in order to invoke particular methods on objects in the domain.

An OO-DTE domain is analogous to a CORBASec privilege attribute. In OO-DTE, principals are given access to domains by virtue of the X.509 certificates they hold. OO-DTE types are similar to CORBASec rights, where the rights family consists of the entire collection of types declared in DTEL++ statements. To be more precise, each OO-DTE type would be associated with two different CORBASec rights: one for *invoke*, and another for *implement*. OO-DTE domain definitions, which map domains to accessible types, are analogous to a CORBASec access policy, which maps privilege attributes to rights. OO-DTE Type assignment statements are analogous to CORBASec Required Rights because they bind access requirements onto methods.

7.2 OO-DTE Types vs. Standard Rights Family

While the standard rights family (get, set, manage) is simple and general, it's usefulness is limited. Consider a requirement for protecting the `Book` interface in the library application illustrated previously: patrons and librarians are allowed to reserve books, but only librarians are allowed to check out books. Both of the `Book` methods `reserve` and `checkOut` change the application's state. Hence ,for both, the most appropriate required right from the standard rights family is "set". But if the "set" right is granted to both librarians and patrons, patrons will be allowed, inappropriately, to invoke the `checkOut` methods. On the other hand, if only librarians are granted the "set" right, then patrons will be prevented, inappropriately, from invoking the `reserve` method. Unfortunately, the standard rights family cannot readily address these simple requirements.

The underlying issue is that the interface to an object often includes multiple "set" (or "get") methods whose sensitivities vary greatly. To adequately specify the required rights for the interface, a

[27] Note that CORBASec Security Policy Domains are not related to OO-DTE Domains.

distinct right might be needed for each method. The optimal number of these distinct rights depends on the characteristics of the application. The standard rights family does not provide this granularity or flexibility. By contrast, the "rights family" for a system using OO-DTE is simply the collection of OO-DTE types declared in DTEL++ statements. The simplicity of the OO-DTE solution for the library application, presented in Section 3 above, illustrates OO-DTE's flexibility and effectiveness at solving this common problem.

7.3 Per-Object Access Control

OO-DTE provides named objects and type templates so that access control can be configured on a per-object basis when needed. CORBASec states, however, that "Required Rights are characteristics of interfaces, *not* instances. All instances of an interface, therefore, will always have the same Required Rights."[28] Consequently, it is not possible to directly specify per-object access control using the Required Rights mechanism. To get around this restriction, the application architect is forced to put individual objects or collections of objects – all of which belong to the same interface -- into separate Security Policy Domains. The specification defines a Security Policy Domain as "a set of objects to which a security policy applies for a set of security related activities and is administered by a security authority."[29]

Consider the use of per-object access controls to provide special handling of antique books in our library example as presented in Section 3. Addressing this handling requirement by using the CORBASec Required Rights necessitates placing antiques into a different Security Policy Domain than ordinary books. This seems awkward and artificial because both kinds of books are objects "to which a [single] security policy applies" and both are probably controlled and "administered by a [single] security authority". Moreover, the administrative and programming ramifications of using multiple Security Policy Domains are unknown. CORBASec is deliberately silent on many security management issues including how the Policy Domains are created, deleted, and configured in relationship to one another and how objects are moved between domains. For simple systems like our library application, forcing designers to use multiple Policy Domains may ultimately prove cumbersome, and unnecessary given that OO-DTE achieves the desired result without them.

7.4 Supporting and Exploiting Inheritance

DTEL++ uses a collection of wild-card rules to facilitate the assignment of types to methods and make these assignments as compact and intuitive as possible. In particular, DTEL++ exploits the inheritance hierarchy. When types are assigned to the methods of a base interface, by default, the inherited methods in all derived interfaces automatically inherit those type assignments. As a result, when derived interfaces are added to an application, few if any additional DTEL++ statements may need to be written, even if the base interface was complex and required numerous DTEL++ statements. In addition, DTEL++ provides a variety of optional flags to control and selectively override the propagation of type assignments through the inheritance hierarchy.

In the examples in CORBASec, Required Rights are shown as a table that enumerates the rights required for each method in *each interface*. Inheritance is not discussed. This suggests that when a derived interface is defined, it may be necessary to enumerate the required rights for each inherited method … even if they are identical to the rights for the base interface's methods. This will be burdensome when deriving new interfaces from lengthy base interfaces. Furthermore, it is clearly unnecessary, as DTEL++ illustrates, and is conducive to errors and inconsistencies. The specification doesn't prohibit an implementation from exploiting inheritance to make the enumeration of rights more compact. But, it also fails to provide guidance or insights into the benefits, issues, or features that might be required.

[28] Section 15.6.4, "Access Policies"
[29] Section 15.3.8, "Domains"

8. Observations

In the testing of our OO-DTE prototypes, most features of the system worked as expected. Some of the features that appeared to be unimportant in the design phase proved quite useful after implementation; other more complex features have so far had limited value.

OO-DTE has been implemented in ILU on BSD/OS and Orbix on Solaris. Both versions were successfully demonstrated with moderately complex demonstration applications. The applications included three to four clients and a similar number of servers. In addition, OO-DTE was partially integrated into the JFACC Enhanced Planning Service, a large planning program developed under DARPA funding.

In practice, DTEL++ has proven to be simple, flexible, and easy to use as a policy specification language. DTEL++ policies tend to be compact, usually no larger than the related IDL file; in many cases they will probably be much smaller. This is largely due to the automatic inheritance of assigned type bindings and the ability to have default type bindings for an entire interface or module. Also, the rules in a DTEL++ policy are concise and straightforward. The reduction of subjects and objects to equivalence classes (*domains* and *types*), and of modes of access to just *invoke* and *implement* greatly simplifies access control decisions and makes the policy easier to understand and to write.

Per-object access control via type templates has been very valuable. It has eliminated the need for some security-aware application code. Per-object access control is particularly useful when securing standard services, such as Naming, that large user communities will share.

Automated policy distribution proved invaluable during testing to distribute policy changes quickly to multiple hosts, including the ORB Gateway.

A few aspects of OO-DTE did not work as well as hoped. The syntax for defining type templates and the inheritance relationships between type templates is confusing and needs refinement. Among other changes, we plan to experiment with a syntax that allows multiple interfaces per type template. We have not used inheritance in type templates extensively, possibly due to the additional complexity of the language.

Rules that were easily expressed in the specification for DTEL++ have been difficult to implement in the compiler. Specifically, rules governing the assignment of types in interfaces with multiple inheritance have proven problematic. In this case, it is possible that no matter what rule is selected, the behavior may seem complicated or confusing to policy writers. One possibility is to have no default in cases that could be ambiguous, and instead insist on an explicit type assignment.

8.1 Issues

There are several issues remaining to be resolved in the implementation of OO-DTE:

- Delegation – SSL does not currently support delegation. Intermediate servers must be given more privileges than desirable in order to perform invocations on target servers.
- Object naming – Modified ‘markers’ (also known as ‘instance handles’ and ‘keys’) are not portable across ORB products. A portable, interoperable mechanism for transporting object names *in the IOR* is required.
- Inadequate interceptors – Commercial ORBs do not provide consistent and adequate access to required interfaces, such as IOR construction and Service Contexts. This may be partially rectified by implementation of the Portable Interceptors RFP, currently under consideration in the OMG.

- Lack of interoperability – Orbix and Visibroker[30] were not able to interoperate using IIOP across SSL. Neither ORB supports connections between non-SSL clients and SSL-enabled servers. OO-DTE allows unauthenticated clients to be placed into weak domains by the server, but this feature cannot be implemented in current ORB products.

8.2 Future Plans

OO-DTE research and development is continuing under DARPA funding. We plan to continue research in the following areas:

- Attribute certificates - The current implementation places the domain privilege attribute in the Organizational Unit of the certificate. Privilege attributes should be separated from identity and stored in an attribute certificate.
- Hierarchical policy modules - There is a need to distribute policy throughout large organizations, crossing departmental boundaries. Most departments do not need all of the policy, but need enough of the policy to ensure consistency throughout the organization and interoperability between departments.
- Java RMI – We are extending OO-DTE for use with Java RMI. One issue that has surfaced is that RMI supports overloaded method names. Because CORBA IDL does not, neither does DTEL++. DTEL++ will need to be extended accordingly.
- Role instances – In some applications there is a need to distinguish among different instances of the same role and extend different rights to each. For example, in a medical records application, there may be several instances of the primary physician role. Each instance should have access to the same fields in patient records, but for different patients. We plan to extend OO-DTE to support such requirements.

9. Summary

Although distributed object technology has matured greatly during recent years, the lack of practical, integrated security mechanisms, particularly for access control, remains an obstacle to its deployment in many application domains. It is often necessary to control access to individual objects and methods. In large systems, however, these can be so numerous that the resulting proliferation of access control information can be overwhelming. We have described Object Oriented Domain and Type Enforcement, a research technology for organizing, specifying, and enforcing access control that has been prototyped and integrated with a commercial ORB and SSL. OO-DTE is an outgrowth of our earlier research into access controls for secure operating systems.

Our experience developing OO-DTE suggests that it has been successful in meeting its original goals:

- Object-oriented – OO-DTE is object oriented. DTEL++, its compilable policy language, resembles CORBA IDL, supports OO abstractions, and takes advantage of the inheritance hierarchy to improve the conciseness and understandability of access control policies.
- Scalability and manageability – DTEL++ provides wild-card techniques for assigning OO-DTE types to methods based on the inheritance, lexical name scoping, and object naming hierarchies. These techniques allow a few DTEL++ statements to control thousands of objects and methods. Mechanisms are also provided to automatically distribute policy changes to large numbers of clients and servers.
- Fine-grained control – OO-DTE provides fine-grained control over individual objects via named type templates. Type templates allow distinct combinations of OO-DTE types to be assigned to individual objects or collections of objects whose names share subtrees of the object name space.
- Role-based access control – OO-DTE provides access control based on user roles. Roles, implemented as OO-DTE domains, can be defined in an application-specific manner via DTEL++. A user's authority to act in a role is represented by the value of a field in the user's X.509 certificate.

[30] Visibroker is a product of Inprise Corporation

- Compatibility with commercial products – OO-DTE has been implemented as a plug-in module for Iona ORBIX and SSL and is being adapted for Inprise Visibroker.
- Transparency – The OO-DTE plug-in supports security-unaware applications by automatically invoking SSL authentication and access checks without direct participation by application code.

We have compared OO-DTE to the access control terminology and features of CORBASec and described what we believe are important advantages of OO-DTE over CORBASec's Required Rights and standard rights family. These advantages include the ability to 1) address a wider range of real-world access control requirements; 2) provide per-object access control within a single Security Policy Domain; and 3) express policies for derived classes in a more compact and understandable manner.

We have also identified a number of areas where OO-DTE could be improved, including the syntax for type templates and rules for assigning types in interfaces that use multiple inheritance. Plans for the future include extending OO-DTE for use with Java RMI, supporting role instances, providing hierarchical policy distribution, and carrying role authorization information in attribute certificates rather than identity certificates. Improvements and extensions of these kinds, combined with OO-DTE's designed-in scalability, should allow OO-DTE to mature over time from research prototypes to a practical technology base suitable for incorporation in commercial products and deployment in large-scale distributed object systems.

Bibliography

[BADG] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, S. A. Haghighat, *"Practical Domain and Type Enforcement for UNIX,"* Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, CA, May 1995

[BALD] R. W. Baldwin, "*Naming and Grouping Privileges to Simplify Security Management in Large Databases"*, Proceedings of the IEEE Symposium on Security and Privacy, page 116 (May 1990).

[BOEB] W. E. Boebert and R. Y. Kain, *"A Practical Alternative to Hierarchical Integrity Policies,"* Proceedings of the 8th National Computer Security Conference, pp. 18-27, Gaithersburg, MD, September 1985

[CORB] CORBA Services: Common Object Services Specification, Chapter 15, Security Service Specification, November 1997 Object Management Group, Inc.

[DENG] R. H. Deng, S. K. Bohnsle, W. Wang, A. A. Lazar, *"Integrating Security in CORBA-Based Object Architectures"*, 1995 IEEE Symposium on Security and Privacy, page 50

[DOWN] D. Downs, J. Rub, K. Kung, C. Jordan, *"Issues in Discretionary Access Control"*, 1985 IEEE Proceedings of the Symposium on Security and Privacy, page 208

[HU] W. Hu, DCE Security Programming, 1995, O'Reilly & Associates, Inc.

[KUHN] D. Kuhn, J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrila, *"Role-Based Access Control for the World Wide Web"*, 20th National Information Systems Security Conference, page 331

[NICO] V. Nicomette and Y. Deswarte, *"An Authorization Scheme For Distributed Object Systems"*, 1997 IEEE Symposium on Security and Privacy, page 21

[OSG] Orbix Security Guide, 1991, IONA Technologies PLC.

[SALT] J. Saltzer and M. Schroeder, *"The Protection of Information in Computer Systems"*, IEEE Proceedings, 63(9), March 1975

[SHER] D. Sherman, D. Sterne, L. Badger, S. Murphy, K. Walker, S. Haghighat, *"Controlling Network Communication with Domain and Type Enforcement"*, 18th National Information Systems Security Conference, October 1995, page 211

[TALL] G. Tally, D. F. Sterne, C. D. McDonell, “Sigma Project: DTEL++ Language Specification”, Trusted Information Systems, September 1998

Architecture and Design for Secure Virtual Enclaves

by Terry Benzel, John Sebes, and Richard Yee

1. Introduction: Collaboration and Security

Use of computers and networking to share information is a widespread part of life in our technological culture. Common sense about this common fact tells us that when we share something important, we also want protect it. If to share something is also to put it significantly at risk, then our sharing is limited to items we can afford to lose. This limitation is analogous to the state of computer networking technology today. Internet technology is increasingly used to provide networking and services for collaboration between enterprises via sharing information and application services. The power and flexibility of sharing techniques is increasing, but protection techniques lag far behind. Security technology— the collection of such protection techniques— instead of enabling sophisticated collaborations, is often a bar to safe sharing. As a result, sharing is either insufficient to the needs of collaborations, or information is shared with an inappropriate degree of risk.

The results of these limitations of current technology and practice present the challenges for creating a security infrastructure for "virtual enclaves." We use the term virtual enclave to refer to a situation in which multiple collaborating enterprises are sharing information and services that require protection in order for the collaboration to be successful. Providing security for virtual enclaves means applying security technology to the kinds of sharing enabled with Internet technology, in order to provide appropriate protection for the shared information and services.

1.1 The Internet as Basic Infrastructure

Internet technology is now a critical part of our society's infrastructure. Internet technology—IP networking and the myriad of network, application, and communication services built on top of it—has become prevalent so rapidly that it is viewed as a recent innovation in most of contexts where it is in wide usage. It has become almost trite to describe the Internet as a fixture that is as central in some ways as highways, airways, telephones, banks, hospitals, and all the other everyday fixtures of the "systems" we all rely on. In fact, all these systems rely in some measure on Internet technology. The transportation system, the telecommunication system, the financial sector, the healthcare system-- each of these relies on computers and networks for daily operation. At an ever-increasing pace, Internet technology is on the rise as the central technology for networking and application services for these systems.

Partly as a result of this rise, and partly as a driver of it, is the use of Internet technology to interconnect organizations and their information technology (IT) systems. Not long ago, the most common use of this connection was to enable simple sharing of information with the world large. The term "home page" entered the popular lexicon. Slightly less prevalent but more significant, organizations became interconnected within themselves, using the power of Internet technology to share information broadly within the organization. The term "intranets" entered common usage, enabling one to sound shrewd in conversation around the water cooler by saying that intranets are a much more significant market than the Internet.

More recently, organizations are increasingly interconnecting their intranets to those of other organizations. The term "extranet" may not yet be found as frequently in the pages of IT trade magazines as "Internet" and "intranets," but its frequency is increasing. However, inter-organizational inter-networking is more significant, and in some ways more subtle, than the common usage of "extranet" implies. One should not minimize the technological challenges of sharing low-risk information with customers, suppliers, or contractors. Low-risk, in this context, means information that is fundamentally about the sharing party (e.g., status of a customer backorder) and that would not cause significant harm if publicly disclosed. Nevertheless, such activities are only the simple leading edge of what we refer to as collaborative computing.

1.2 Collaborative Computing

Collaborative computing is the sharing of information and services between organizations that collaborate, i.e., have some common mission, goal, or activity. The various forms of collaboration certainly include customer and supplier interactions. More generally, however, collaborative computing spans interactions of many sorts in many situations with many kinds of organizations. The interactions can be based on limited trust between collaborating organizations, little or no prior knowledge of collaborators, short-term and/or evolving collaborative agreements.

The information and services shared are often not directly related to collaborators, pre-date the collaboration, and may have significant value and wide use independent of the collaboration. Information shared by an organization with some collaborators may be a subset of some information base usually reserved for internal use. Shared services may include only limited capabilities of some service usually reserved for internal use. In other words, collaborative computing differs from many kinds of current extranets in that the information and services are shared as an exception to the rule "reserved for internal use by authorized employees only." As a result, collaborative computing creates requires security mechanisms that enable important information and services to be shared in a carefully controlled manner: only authorized collaborators gain access, and their access is limited only what is required for the collaboration.

Examples of collaborative computing can be seen now in organizations on the leading edge of technology usage and collaborative practices. In military environments, joint task forces (JTFs) share information and applications for distributed collaborative planning. Not all information pertinent to a unit in a JTF is shared, but all the information about the unit's role in the JTF is shared. Similar situations exist in civilian settings in disaster or incident response teams, where various government organizations and corporate units rapidly form a team. Disparate government units (e.g., law enforcement and civil aviation) shared information in a limited way that is beyond sharing in ordinary settings. Corporate units similarly share information with outside organizations (e.g. disclosing cargo manifests) without allowing general access to sensitive corporate data. In purely commercial settings, there is a range of collaborative sharing practices. At one end of the range is sharing at leading edge of extranet practice, where a company allows contractors or outsourcing organizations to have authenticated controlled access to data sets specifically pertinent to their function for the company. At the other end of the spectrum, partners in multi-company projects are allowed access to corporate data via the Internet, using access controls and authentication features of application servers. In some setting, such as the aerospace industry, partners in a project are often competitors.

1.3 Challenges and Opportunities

As all these examples show, collaborative computing is not limited to sharing the kind of low-risk information common to current extranets. As a result, strong security measures are required to provide a high degree of flexibility in control of sharing. Developing security infrastructure to meet these needs is a challenge, but one for which the time for solutions has come. Many elements of virtual enclave security exist already. Some are not, but are in reach. Development of new elements and integration of existing ones can yield the enabling security technology for a new level of safe collaborative computing.

2. Secure Virtual Enclaves

To meet the challenges and enable the opportunities of secure collaborative computing, the Secure Virtual Enclaves (SVE) project is pursuing one overarching goal: development of technology that enables multiple enterprises to engage in controlled collaborative computing using distributed applications and open networks.

2.1 SVE Project Goal

Each of the key words in this SVE project "mission statement" deserves a bit of elaboration. First, an enterprise is an organization that uses its IT system (or systems) to interact with other organizations, according to management decisions about appropriate interaction. An enterprise may be a single enclave, in the terminology of the SVE project, or it may consist of multiple enclaves. The key aspect of enterprises is that it is at the enterprise level that decisions about collaborations are made, or ratified, or delegated. For example, an aerospace contractor company is an enterprise, and the decision to have a Web site for public consumption was made at the enterprise level. Some delegation might be performed with the decision to allow organizations within the enterprise, e.g. the fighter plane division, to have their own public Web site. If the division were to engage in a new collaboration-- such as allowing project sub-contractors to access project-private data via the Internet-- the decision to do so might be made at some part at the enterprise level. Nevertheless, the division's IT system would be the enclave that interacts with contractor enclaves to form a virtual enclave.

A second key word is control. The goal of the project is to enable collaborative computing by providing sufficient controls that information and services can be shared with collaborators without undue risk. The required level of control can be achieved by the combination of two kinds of activities: technological, by the application of security technology to the mechanisms for sharing; administrative, by specifying exactly what is to be shared with who. Of course, the security technology needs to support the administrative activities, so that a suitably fine degree of control can be achieved. For example, a single application service in an enclave may manage a wide of resources. All resources are available for use via the application within the enterprise, but only a subset is to be shared with collaborators in one particular SVE. Both the technology and the administration must be able to support the precise specification of subset, and also a clear definition of the SVE collaborators who are allowed access.

Another key word is open networks. The SVE project is focused on security for collaboration using Internet technology and practice. This focus includes collaboration via the Internet, where sharing with collaborators must be enabled without exposing assets to undue risk of access by the world at large. The focus also includes other IP-based networks which collaborators share with non-collaborators. Hence, the notion of open networks includes both the use of standard inter-networking technology, and also using networks which are open to use by other enterprises with which one does not wish to share with.

A final key word is distributed applications. Standard inter-networking technology provides the foundation for collaborative computing, and other elements of Internet infrastructure and basic services (e.g. DNS, SMTP, FTP) provide basic building blocks. Many of an SVE's shared assets will be built on top of this basis using a range of distributed application technologies, including combinations of them. These distributed application technologies include the World Wide Web, CORBA, Java, ActiveX/OLE/COM, and combinations with legacy applications that may be wrapped or front-ended by these technologies.

2.2 Objectives and Challenges

There are several objectives for the way in which the SVE project should meet its basic goals and challenges that go along with those objectives.

One objective is for SVE security techniques to apply to a variety of distributed application technologies, as well as common network services such as FTP that may not be rightly called distributed applications, but are important tools for collaboration. Similar variety exists in the area of distributed security technologies. Various technologies (such as SSL, Kerberos, DCE security, SPKM, S/MIME) are used by different distributed application technologies. SVE security techniques should encompass the use of multiple distributed security technologies as well as distributed application technologies. This objective is inherently challenging not only in terms of scope but also because of historical difficulties in providing effective and manageable security mechanisms at the application level. Recent experience with distributed applications and security indicates that middleware and security technologies have reach sufficient integration where some of these difficulties may be removed. The challenge for SVE is to extend this progress into the arena of inter-enterprise computing, where there is much less experience in effective integration of security. A final challenge in this area is simply that the breadth of application and security technologies requires a single security framework for effective collaboration. While a truly all-encompassing framework may be out of reach for some time, the more limited challenge for SVE is to develop a framework for SVE-specific security, and demonstrate the inclusion in that framework of multiple application and security technologies.

A second area of objectives is a focus on existing computing and networking infrastructure. All SVE solution are to be based on commodity, off-the-shelf operating systems (e.g. Solaris, NT), middleware (e.g. CORBA, COM), protocols (HTTP, IIOP), etc. No SVE solutions are to be based on modifications to operating systems, middleware, protocols, or any of the elements of infrastructure that distributed applications rely on. The same applies to existing security infrastructure (e.g. elements of public key infrastructure) and protocols. Likewise, no SVE solutions are to require any changes to current practices of open networking and existing network technologies, or to rely on new features of network infrastructure. The challenge inherent in these objectives is simply that SVE goals have not been met within these constraints, although there is a body of related work in which some of these constraints have been relaxed, e.g. modifying the OS, or assuming a common security model and authentication infrastructure across enterprises. As with the first area of challenges, a complete overarching solution may remain out of reach for some time, but now is the time when it is feasible to develop a sound and flexible security foundation for COTS-based collaborative computing.

Part of our approach to mitigating the risk of challenges in both these areas is the assumption that collaborating enterprises have some knowledge of one another and some limited trust in one another. SVEs are formed not by total strangers, but limited partners that share some goal or mission. In pursuit of that mission, each partner is willing to share information and to trust other partners discretion in using it. This trust and this willingness is strictly limited, however, and strong security measures are needed not only to keep out non-collaborators, but also to strictly limit the capabilities of legitimate collaborators. Providing these security measures, and integrating them into the distributed applications used for collaboration, is the main goal of the SVE project. Besides providing a project scope that is challenging but manageable, this focus also provides the possibility for meeting some of the scalability and adaptivity requirements that are the topic of the next section.

3. Definitions and Requirements

Before describing our development of SVEs, we first describe some specific requirements for SVE security, and define the terms in which the requirements are stated. Previous sections have defined enterprises, and collaborative computing. Secure collaborative computing is when sharing between enterprises includes sharing resources that are not accessible to the world at large, but only to specifically authorized collaborators. We use the term export to mean the action where some set of resources internal to one enclave is defined to be available to some set of collaborators. This is one of the critical steps of control in SVEs. Only an enclave itself can define which resources are exported from the enclave. Collaborators can access only those of an enclave’s resources that have been exported. We use the term policy to refer to a set of rules about which collaborators may access which resources. Likewise, to control is to enforce a policy, i.e. to ensure that collaborators get no more access than is intended, and to ensure that only authorized collaborators get any access at all to exported resources.

Note that the term export is used in a virtual sense. That is, resources shared with collaborators are not exported in the sense of being physically moved or copied to collaborators. Rather, shared resources are exported to the abstraction of the secure virtual enclave, which represents the shared resources and collaborators using them. In actuality, exported resources continue to reside inside the enclave, in many cases co-resident on one host with other resources that are not exported.

This distinction is critical to flexibility of SVEs: resources do not have to be reconfigured in order to be shared. This virtual export aspect of virtual enclaves is one of the primary distinctions between SVEs and current techniques of collaborative computing. For example, one enterprise can with another via physical export of some of its information that were previously intended for internal use only. That is, the shared subset of information is copied or replicated to a distinct server host, which is situated in a region of the network that is accessible to outsiders (a service net, or a DMZ). In other words, collaborative access can be established without danger to non-exported resources by high-overhead administrative measures that cannot provide the flexibility and adaptivity of SVEs.

Using these basic definitions, we can sum up a great deal of the technical tasks to be performed on the SVE project: develop facilities for policies to be expressed, disseminated, and updated; develop mechanisms for the enforcement of these policies; integrate enforcement mechanisms with various distributed application and security technologies. With these definitions of terms and tasks, we can more precisely define secure virtual enclaves and the various requirements for them.

3.1 Enclaves: Real, Virtual, Secure

An enclave is a collection of computers and networks managed by the same organization, and subject to the same security policy. Small to medium enterprises may be a single enclave, while large distributed organizations may be segmented sufficiently for sub-organizations to act as distinct enclaves. A virtual enclave is formed when a group of collaborators' real enclaves jointly share services/resources with one another. Such sharing arrangements are based on well-defined and limited trust relationships between the organizations. Each enclave participating in a virtual enclave is a member of the virtual enclave. Each enclave specifies which of its internal resources are exported to a virtual enclave, i.e. are accessible for collaborative use by other members of the virtual enclave.

A secure virtual enclave is a virtual enclave where sharing is controlled so that access to exports is constrained in the following ways. Every member enclave's every export is logically segregated from other resources that are not exported to the same SVE, or any SVE. This logical segregation implies that SVE security mechanisms can permit authorized access to exports, but deny any access to resources that are not exported, even when the exported and non-exported resources are closely related (e.g., located on the same server host, managed by the same application). Each export is protected from access by parties that are not members of the SVE. This protection implies that SVE security mechanisms ensure that sharing with collaborators does not devolve to sharing with the world. Each export is access-controlled to ensure the above segregation and protection.

An SVE security policy states what resources are accessible to whom. Each member enclave of an SVE states a part of the policy, by defining its exports, and by defining for each export an access control rule to be implemented by the SVE security mechanisms in the member enclave. As described in more detail below, other member enclaves can provide recognition rules so that an exporting enclave can recognize which principals of other enclaves are governed by the access rule for a given export. The export definitions, access rules, and recognition rules from all member enclaves together comprise the definition of an SVE's security policy.

Given these SVE definitions, we can state the security functional requirements for the SVE security mechanisms that enforce SVE security policies. First, each member enclave of an SVE enforces its own controls on access to its own resources exported to the SVE. Thus, each enclave enforces the part of an SVE security policy that describes permitted access to the resources exported by the enclave.

Second, the definition of exports must be fine-grained, in order to allow definition of exports that are subsets of internally available resources. Thus, each enclave's enforcement of its part of an SVE security policy is the enforcement that logically segregates exported and non-exported resources. Finally, overall security policy for an SVE is enforced jointly by all enclaves in the SVE. That is, each enclave's enforcement of its part of an SVE security policy depends in some measure on information from other member enclaves, e.g. recognition rules for member principals.

3.2 Scalability and Policy

In addition to the basic security requirements stated above, there are several requirements for scalability, which is a critical factor for wide-scale utilization of SVE security. As mentioned above, an SVE security policy is the sum of policy elements from member enclaves. Each member submits policy elements that describe the resources that they export to the SVE, and the principals that require access to resources exported by other member enclaves. This division of responsibility is critical not only for each member enclave to retain fundamental control over its resources. The division of labor is also critical for the ability to build scale SVEs up to large scales, both large scaled individual SVEs, and also large scales of numbers of distinct SVEs than an individual enclave can be a member of. As is so often the case, decentralization is critical to scalability.

Another scalability factor is driven by the relationship of an SVE's security policy to the security policies of each of the member enclaves. From an individual enclave's viewpoint, there is a relationship between its overall enclave security policy (including rules on internal access, on general access from the world at large, and on collaborative access) and the policy elements it contributes to SVEs. For example, an enclave will only export some resource to an SVE (thereby creating an element of that SVE's policy) only if the export is consistent with the enclave's overall policy on collaborations. However, there is no close relationship between an enclave's overall policy, and SVE policy elements from other member enclaves. Each enclave need not even be aware of the exports of other member enclaves.

Moreover, each enclave's enforcement of the SVE security policy need not depend on an understanding of the overall policies of other member enclaves. Each enclave need not— and cannot if SVEs are to be scalable—understand other enclaves' authentication and authorization schemes, and need not interpret the security policy data of other enclaves. Instead, each enclave uses recognition rules from other enclaves in order to determine when another enclave's principal should be granted access to an SVE resource. The recognition rule encodes purely syntactic measures for recognizing authentication and/or authorization data of principals. There may be a close semantic relationship between the way some authentication data is used in a recognition rule for SVE access control, and the way that the same data is used for access control with the enclave that stated the recognition rule. But if there is such a relationship, the knowledge of the relationship is not required by the SVE security mechanisms of other enclaves. In other words, lack of semantic linkage between enclave policies is critical for scalability of SVEs. As a consequence, an SVE's operation is based on an SVE policy composed of distinct elements. Each of these policy elements is defined by one enclave but is meaningful to other enclaves without knowledge of the local policy of the defining enclave. More information about these policy elements is provided by the discussion in Section 4.1 of types, domains, and principal recognition rules.

3.3 Scalability and Grouping

Another scalability requirement, also policy-related, concerns the way that SVE security policies are stated and disseminated among member enclaves. Even moderate sized SVEs (a handful of participants, a few exports per each) can contain enough complexity to raise scalability issues. In addition to the multiple member enclaves exporting multiple services, there may be a large number of principals in SVE, and a wide array of security attributes (enclave-specific authentication and/or authorization data) associated with each. Also, each individual export may contain large numbers of individual resources (especially exports of distributed object applications) that need to be recognizable as a group and also distinct from other related resources that are not part of the export. Finally, each enclave may participate in multiple SVEs.

As a result of these facts of life in SVE participation, it is essential that SVE policies (and security runtime data) be formulated in terms of large equivalence classes or groups of accessible resources and accessing principals. Otherwise, even medium-scale SVEs would be governed by policies that would be formulated in ways to complex for most people to manage correctly. And without correct management, even the best security schemes can allow unauthorized access as a result of mis-configuration.

Therefore, each export is defined as a group of resources that are equivalent in the sense that they are all subject to the same single access rule. For two resources to be SVE-accessible in distinct ways, the two resources must be part of distinct exports with distinct access rules. As a result, there can be a manageable number of access rules even if there are large numbers of distinct resources exported from one enclave to an SVE. The same grouping approach must be applied to accessing principals. Access rules should not generally refer to individual principals, but rather to one or more accessor classes. All members of an accessor class are equivalent in that they all have equal access to any exported resource for which access is granted to the accessor class.

As a result, there can be moderate numbers of accessor classes even if there are large numbers of principals. And with moderate numbers of accessor classes in an SVE, individual access rules can be kept to a manageable level of complexity.

A final note on scalability is related to the goal that SVE mechanisms apply to multiple types of distributed application and security technologies. The very variety of SVE access-controlled resources (e.g., Web pages, FTP files, CORBA or COM or Java objects) underscores the need for simple rules that group together similar resources. Each export defines some set of resources for one kind of application, and must also be able to tie that set of resources to some access rule. If one export of Web pages is subject to the same access controls as another export of COM objects, then the two sets must be associated with the same access class governed by the same access rule stating the access permitted to some list of accessor classes. Furthermore, membership in a single accessor class can be established by distributed authentication credentials of multiple different security technologies.

3.4 Adaptivity Requirements

Another significant area of requirements for SVEs is in the area of adaptivity. SVE security policies must by dynamic, i.e. subject to changes that can be frequent and must be put in place rapidly and automatically, without human coordination between enclaves. As a result, SVE security mechanisms must be to enforce a dynamic SVE security policy. Therefore, there must be a facility for the runtime security data to be automatically reconfigured to account for changes in SVE security policy. These changes are the result of SVE administrative activities in individual enclaves. Any of an SVE's member enclaves can add or remove exports, redefine exports (change the definitions of what is in the exported set of resources), and define or redefine the access rights of principals in the enclave that are able to access resources in the SVE. In addition, all of these changes can occur at once as entire enclaves join or leave an SVE.

As a result of any of these changes, one enclave's policy element modification must be propagated to other enclaves in the SVE. These policy modifications, when received, are effected by modification to the security runtime data of SVE enforcement mechanisms in each of the other enclaves. For scalability reasons, there is no central policy service or policy data exchange. Instead, each enclave is responsible for multicasting to the dynamic group of the other enclaves in the SVE.

3.5 Policy Requirements

A final area of requirements addresses the nature of an SVE security policy. Thus far, discussion of the policy has focused on access control, i.e., policies that describe which collaborators are permitted to what resources. In addition to access control, however, the SVE security mechanisms must be flexible enough to implement other policies. One example of such a policy would be a resource consumption limitation. Even though some set of principals in a collaborating enclave may be permitted to access some exported resources, the exporter may wish to limit the ability of the collaborators to monopolize the use of the resources and degrade internal users' service. Tie-ins to quality of service (QoS) mechanisms may be a related issue, i.e. to enforce upper bounds on QoS of collaborators in order to ensure preferential access to priority accessors. Another kind of policy area would be quality of protection, in which a policy would state that collaborators' access to certain resources must be cryptographically protected. Policy could specify different protection requirements for different accessors and/or resources accessed. Definition of each of these policy areas will be the topic of a later report on the SVE policy definition language.

4. Creating SVEs

To create the technology for virtual enclave security, we will develop and integrate software in various areas. We will develop facilities for definition

- SVE policy elements,
- Secure inter-enclave exchange of policy elements, and
- dynamic update of policy enforcement runtime data.

We will develop SVE policy enforcement runtime software, and integrate it with multiple distributed application and distributed security technologies.

4.1 Virtual Enclave Security Technology Elements

One area of SVE technology elements is in the area of SVE security policy exchange. Each enclave has the capability to specify policy elements pertinent to its exports and its principals that may participate in the SVE to access the exports of other enclaves. Whenever one enclave adds or modifies an element of an SVE policy, the policy update is multicast to the other member enclaves of the SVE. The multicasts must utilize a communication security mechanism, in order to insure that policy updates can be authenticated by recipients, so that recipients can ensure that sender are authorized members of the SVE whose policy they are attempting to update.

Another set of technology elements is in the area of policy enforcement. Each SVE member enclave enforces the portion of the SVE policy that pertains to access of the resources exported by that enclave. Each enclave has one or more instances of a policy enforcement component. Each of these components enforces the SVE policy on one or more sets of exported resources. Typically, each component will enforce the policy on a set of distributed applications based on a single kind of distributed application technology and distributed security technology. The runtime configuration of each such component will be modified in two ways: first, by changes in the policy elements of its own enclave (export definitions, and access rules on exports); second, by changes in policy elements from other enclaves (recognition rules which map principals' authentication data to access rights).

In addition to these technology elements of policy definition, update, and enforcement, there is also the policy model, which defines how SVE policies are stated, and which also provides a high-level design of the algorithms implemented by the SVE policy enforcement components. While the policy update techniques are most critical to adaptivity, the policy model is perhaps the single most critical element of SVE technology for scalability. The policy model is centered on the notion of equivalence classes. Each export is defined as an equivalence class of application resources (e.g., Web pages, COM objects) all of which share an access control rule. Each export is assigned a type, which associates an access rule with the export. Multiple exports can have the same type so as to allow exports of different kinds of application resource to be governed by a single rule if needed. Each type is associated with a domain, which is an equivalence class of principals that are permitted to access resources within an SVE. Part of export definition is an association of the export's type with one or more domains, the member principals of which the exporter will permit to access the exported resources.

Domain membership is specified by the other enclaves in the SVE. Each enclave defines recognition rules that map principals' authentication data into domains. These rules allow an enclave to receive and validate authentication credentials, and determine which domain(s) a principal is in by examination of the credentials' contents according to the recognition rule specified by the principal's enclave.

Notes also that type-domain relationships define equivalence classes not only for access but all elements of the policy such as quality of service and protection.

4.2 Example: SVE Formation

The following abstract example illustrates some aspects of the SVE technology elements described above. In the example, there are three enclaves A, B, and C which have already agreed to form and SVE, and which have exchanged base credentials. The credentials allow each enclave's policy management components to authenticate policy updates from other enclaves in the SVE. Following this initial setup, all other modifications to the SVE policy are propagated automatically between the enclaves. The first update to the initially empty policy is an export statement from A to B and C which we can represent symbolically as follows:

1. Export Alpha: Ta -> Dom1 = A1, A2

This export statement announces the availability of a set of resources exported under the name "Alpha". (The contents of the exported resource set are omitted for brevity.) The export statement also announces a new type "Ta" which is the type of "Alpha" and a new domain "Dom1" which is associated with Alpha. The initial population of Dom1 is two principals A1 and A2. (A1 and A2 are shorthand for recognition rules that map each principal into Dom1.)

The domain and type clauses are also used to state other policy elements beyond access control. For example, quality of protection requirements might be intrinsic to each type, or may be applied to an export independent of type. Similarly, service limitation constraints might be intrinsic to a domain, or might be applied to individual domain/type bindings.

The next steps in the formation of the policy are policy updates by enclave B:

2. Dom1 += B3, B4, B5

and enclave C:

3. Dom1 += C6, C7

which add to Dom1 some principals from each enclave. With this initial policy in place, those principals can request access to Alpha:

4. B3 -> Alpha: permitted
5. C1 -> Alpha: denied

Principal B3's access request is granted because it is in Dom1. Principal C1's access is denied because it is not in Dom1. When this situation is later rectified by further extension of the domain by enclave C:

6. C -> Dom1 += C1

a subsequent access attempt is permitted:

7. C1 -> Alpha: permitted

Further policy modified is the export by enclave B of a new service called Beta.

8. B -> Export Beta: Ta

Because Beta is assigned the pre-existing type Ta, all the members of the domain Dom1 associated with Ta automatically are permitted to access Beta. For example:

9. A1 -> Beta: permitted

A1 is permitted to access Beta because A1 was defined as a member of Dom1 in step 1.

4.3 SVE Components

To create and demonstrate the various elements of SVE technology, we will develop a number of software components and prototypes that integrate these components with a variety of middleware and application software.

One set of components centers on a language for stating SVE policies on access control, quality of protection, service limitations, etc. The language itself is for use by SVE administrators (who control enclave-specific elements of SVE policy) and perhaps to some extent users (who may wish to have a view of the policy that shows what resources area available in an SVE and accessible). Language tools for this language perform a variety of functions:

- translation to configuration data for SVE policy enforcement runtime software;
- translation to policy update data which is multicast to other member enclaves of the SVE of the policy which was updated;
- representation in enclave-centric user interfaces which allow administrators to view and modify export definitions, export types, type-domain bindings, and other elements of policy on exported resources;
- representation in authentication-centric user interfaces which allow enclave administrators to view other enclaves' principal recognition rules, and to view and modify principal recognition rules for that enclave's principals.

Another component is of course the policy enforcement run-time software, which is driven by configuration data that is a result both local-enclave policy statements, and remote-enclave policy updates.

Another component is a set of software for the communication of policy updates between member enclaves of an SVE. This component includes a protocol for transmission of updates, and an interface whereby local-enclave policy updates trigger such transmissions. On the receiving side, there is also an interface policy enforcement configuration data, where reception of update data triggers modification of configuration data. This capability meets the important requirement of on-the-fly update of changes in dynamic SVE policies. In addition to the update protocol there are network protocols for secure transmission of policy updates, and facilities to authenticate senders of policy updates and determine whether they are authorized for the policy updates they have sent. There may also be support for multicast of policy updates messages.

A final group of components are those that perform the functions of integration with the various security technologies and various distributed application and technologies (e.g. CORBA, DCOM, Java, WWW, one more network-based application such as FTP) that use them. For each kind of application supported by SVE prototypes, there will be an adaptation module that analyzes requests for access to resources, and determines whether the request is for an export and if so what access type is associated with the requested resource. For each security technology used by these applications (e.g., SSL, Kerberos/DCE, S/MIME, SPKM) there will be an adaptation module that handles the credentials of that security technology, and feeds authentication data into a common engine that applies the principal recognition rules to information extracted from the credential. Some interplay between these two adaptation modules may be needed, because the same resources could be exported in different ways to multiple distinct SVEs in which a given enclave may be participating.

A very important function of these adaptation modules is to provide at least some support for extensibility to other application/security technologies not integrated with in our prototypes, and to provide strong separation between technology-specific software and more general SVE software. The result should be prototypes that demonstrate that SVE security technology can work with multiple forms of middleware and of security credentials.

5. Architecture and Design Issues

Most of the software architecture and design issues for the SVE project are related to deployment of new SVE components and integration with various applications, distributed application technologies, middleware, and security technologies. This section describes various architectural approaches to these issues, and the implications of these approaches for software design.

5.1 Gateways or End-Systems

There are two basic architectural alternatives for the placement of SVE security mechanisms: at the network boundary, intercepting application requests at the level of network protocol messages; at the end system, intercepting application requests in middleware before they reach the application proper. In cases where there is no middleware, or where the middleware does not support such interception, a variation on the second approach relies on security-aware applications that call out to SVE policy enforcement mechanisms before implementing any application requests.

Both are approaches are of interest, and within project scope. There are significant differences between the two approaches, and implications— sometimes significant ones—for distributed system security. Nevertheless, both approaches have their use in practical systems, and it is important that SVE mechanisms be applicable in both cases. However, the distinctions should be noted. The primary distinction is the often dogmatically discussed difference between perimeter security and security-in-depth.

The security-in-depth argument against end-system security hinges on the often low degree of assurance of externally-oriented security mechanisms deployed on end systems so that the end systems are protected by local security mechanisms rather than network boundary devices like firewalls. In order for end-system security mechanisms to be effective, they must be correct and up-to-date. If even one of them is mis-configured (or configured with a formerly correct but out of date configuration) then it represents a vulnerability which threatens the entire IT system if the end-system is on an enclave’s internal network. If the controls on outside access to local data are ineffective, then it is not only the data requested that it at risk. Other internal-use-only data accessible from the host may be at risk as well. When large numbers of end-systems are involved, and/or when some of the end-systems are not subject to strong administrative control, the risk and the high cost of successful attack argue strongly against a reliance on end-system security.

However, we take the view that end-system security can be an appropriate measure when the number of such end systems (those which are permitted to interact with the outside world to serve collaborators) is reasonably small and all are subject to close and coordinated administrative control. In such cases, the usual security-in-depth risks are significantly mitigated. As a result, there can be significant opportunities to reap performance and efficiency benefits by allowing security responsibilities to be delegated from boundary security components that can be heavily utilized and which become bottlenecks to collaborator access.

Furthermore, SVE security mechanisms themselves have properties that further mitigate security-in-depth risks. The externally-oriented security mechanism—the SVE policy enforcement runtime—is governed by dynamic policy data, the changes to which are supported in large measure by automation via interfaces to both local-enclave SVE-administration components and components that receive and validate policy updates from other enclaves. In our view, these expected benefits of

SVE technology will further strengthen the prudence of judicious employment of end-system security.

Nevertheless, complete reliance on end-system security would not be an adequate approach to SVEs. Some degrees of sharing may simply not be feasible with the proviso that only a small number of tightly controlled internal hosts be involved. Boundary security is also required to handle cases where the end-systems handling the shared resources are not systems on which SVE security mechanisms can prudently be placed. Pragmatic technical considerations play a role also. There may be applications or middleware which simply do not easily support the insertion of end-system SVE security components, either in middleware (where applicable) or via modification of applications to be security-aware. These cases also argue strongly for the capability to place SVE security components at the perimeter of enclaves networks, using boundary security controllers.

5.2 Common Functionality

It is also important to be aware of the elements of SVE functionality that common to both approaches. In both approaches, distributed security technology is used to convey and validate credentials that may authenticate requestors. Authentication data is used to attempt to identify principals permitted to access SVE resources. For such principals, requests are permitted only if the SVE security policy allows them. Policy enforcement software determines domains and types and associated policy on access, quality of protection, etc. This software operates on a base of configuration data that is subject to change either by changes in the local enclave's exports, or by policy updates from other enclaves.

All this functionality is in common, and will be packaged so that it can be deployed either at end-systems or gateways, and integrated with application and security technology adaptation modules. In terms of the SVE project goals, we will demonstrate both approaches, according to priorities large driven by practicality. If one distributed application technology lends itself better to one approach or the other, we will prototype the easier approach. In at least one case, we will prototype both approaches for one security technology and distributed application technology pair.

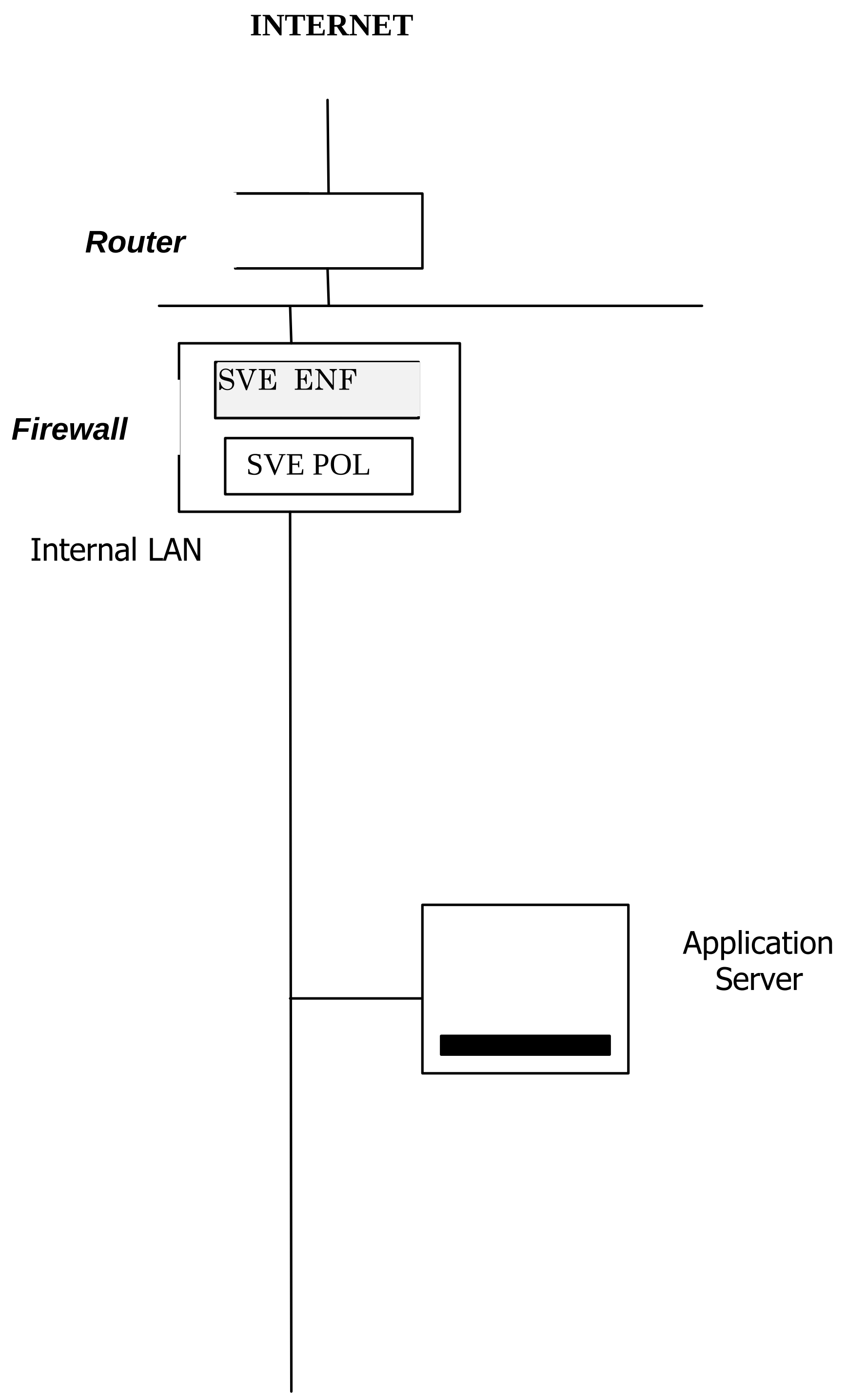

Figure 1: SVE Deployed in Firewall

5.3 Relationship with Firewalls

In current security technology, firewalls are the quintessential boundary controller or network perimeter security component. SVE security mechanisms will necessarily exist in networks that use firewalls for protection. However, SVE security mechanisms need not be implemented in firewalls in order to offer network perimeter protections, although in some cases this may be desirable.

Figure 1 shows a very simple network architecture in which access to an internal end-system is governed by a firewall component that includes an SVE component. Traffic from the outside to the inside application host is allowed only when the SVE enforcement component (labeled "SVE ENF") determines that the requester is an authorized collaborator. Once a permitted request reaches the application server host, access the exported resource (shown in black) proceeds without requiring involvement from any further SVE component.

The internal host may be a file server that is typically for internal use only, but which is outfitted with an FTP server so that collaborators can upload files. The firewall component may be an FTP proxy that invokes the SVE component on requests to the "collaborators FTP server." The SVE component tells the firewall whether a given FTP request for that host is by an authenticated collaborator using local resources that are authorized for use by collaborators. An even more firewall-oriented technique would be for use with very coarse-granularity sharing. In these cases, the enclave would share with collaborators entire server hosts dedicated to applications shared with collaborators. So long as TCP-level access to these hosts can be authenticated (with IPsec or SSL) to be from collaborators authorized to use the host, higher-level protocol handling would be unneeded.

As this example shows, there may be meaningful hybrid approaches of SVE technology use and current firewall technology. However, any SVE technique that could be deployed on a conventional firewall, could also be deployed off of one. And several uses of SVE technology might not be feasibly deployed on a firewall, do to limitations of current firewall technology, difficulty of migration of new security technology to commercial firewalls, and related factors.

Figure 2 shows a similar but alternative technique to firewall integration. Network boundary devices incorporating SVE technology can be deployed next to firewalls at a network perimeter. A router at the junction with an open network is configured to route traffic to the SVE device when traffic is bound for hosts for which outside access is permitted by SVE rules. The device performs firewall-like protocol handling combined with implementation of logic that implements potentially sophisticated SVE policy rules and enforcement, including distributed authentication technology that is not common with current firewalls.

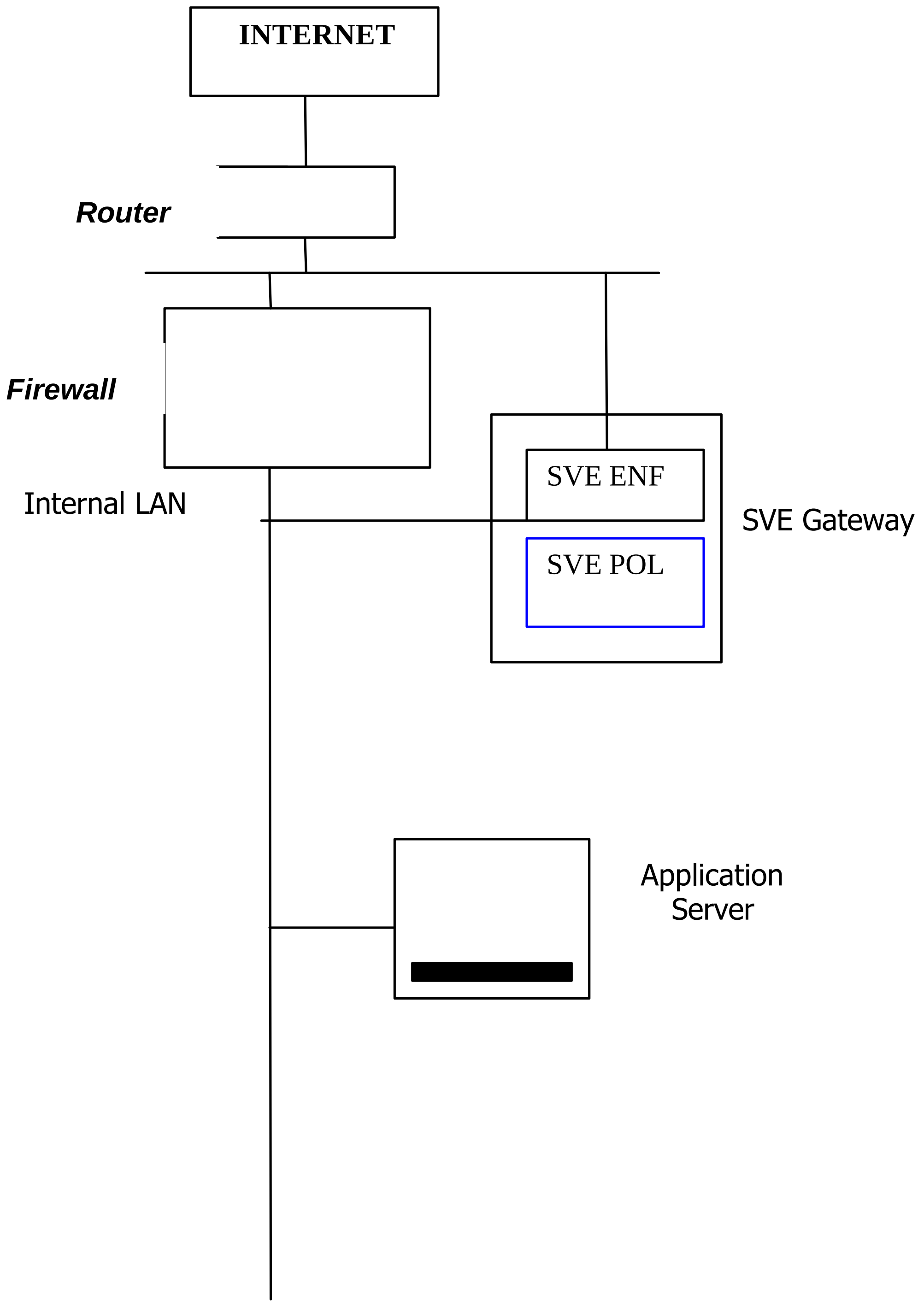

Figure 2: SVE Deployed at Separate Gateway Host

Careful policy management is required in this approach. It is highly desirable with current technology to avoid situations in which an enclave might wish to perform a new export which involved a host for which the router sent traffic to the firewall rather than the SVE device (or one of several SVE devices if needed). Dynamic update of the router configuration would be required in such cases. Although current DARPA efforts (such as AITS RA) include work that could facilitate authenticated secure dynamic remote security management of routers, it would also be desirable to avoid requiring such new technology.

Fortunately, a simple pragmatic approach applies to this situation. We can reasonably expect organizations to have reasonably static (i.e. manageable by manual means with existing technology) policies about what typically-internal-use-only resources might potentially be sharable with collaborators whether or not any current SVE requires access to them. For this set of "potentially sharable hosts" the router would send the traffic to the SVE component rather than the firewall. In cases where no outside access is permitted, the SVE component performs the same primitive blocking functionality of firewalls: block all traffic bound for internal hosts to which no external access is permitted. But when policy change permits access to such a host, the SVE component can accept connections for the host, and perform its usual SVE function of determining whether individual requests are permitted.

Naturally, the approaches of both Figures 1 and 2 can be employed simultaneously if needed. The primary change needed to do so is shown in Figure 3. Since firewall and SVE component both gateway traffic to the inside network, and both contain SVE policy enforcement software (labeled SVE ENF), both require updates from an SVE component that receives policy updates both from other systems and the from local-enclave administrative tools. Rather than having the policy update component (labeled SVE POL) co-located with either SVE enforcement component, it may be advantageous to place the SVE POL component on a third system with a capability for remote distribution of policy updates. Either the firewall or the SVE Gateway must permit policy interchange traffic to the policy distributor.

5.4 Relationship to End-Systems

Figure 4 shows another network architecture, in which some end systems are permitted to communicate with the outside world, and to directly enforce SVE security policy on outside requests. In this case, outside access does not involve an SVE boundary controller, although a firewall is involved. The role of the firewall is to permit outside access to the inside hosts that SVE-capable in order to restrict access to outsiders. The firewall may provide little protection for the inside hosts (e.g. permitting direct routing from outside hosts to the SVE-capable inside hosts) or may provide a first line of defense by limiting outside communication to specific ports and/or protocols.

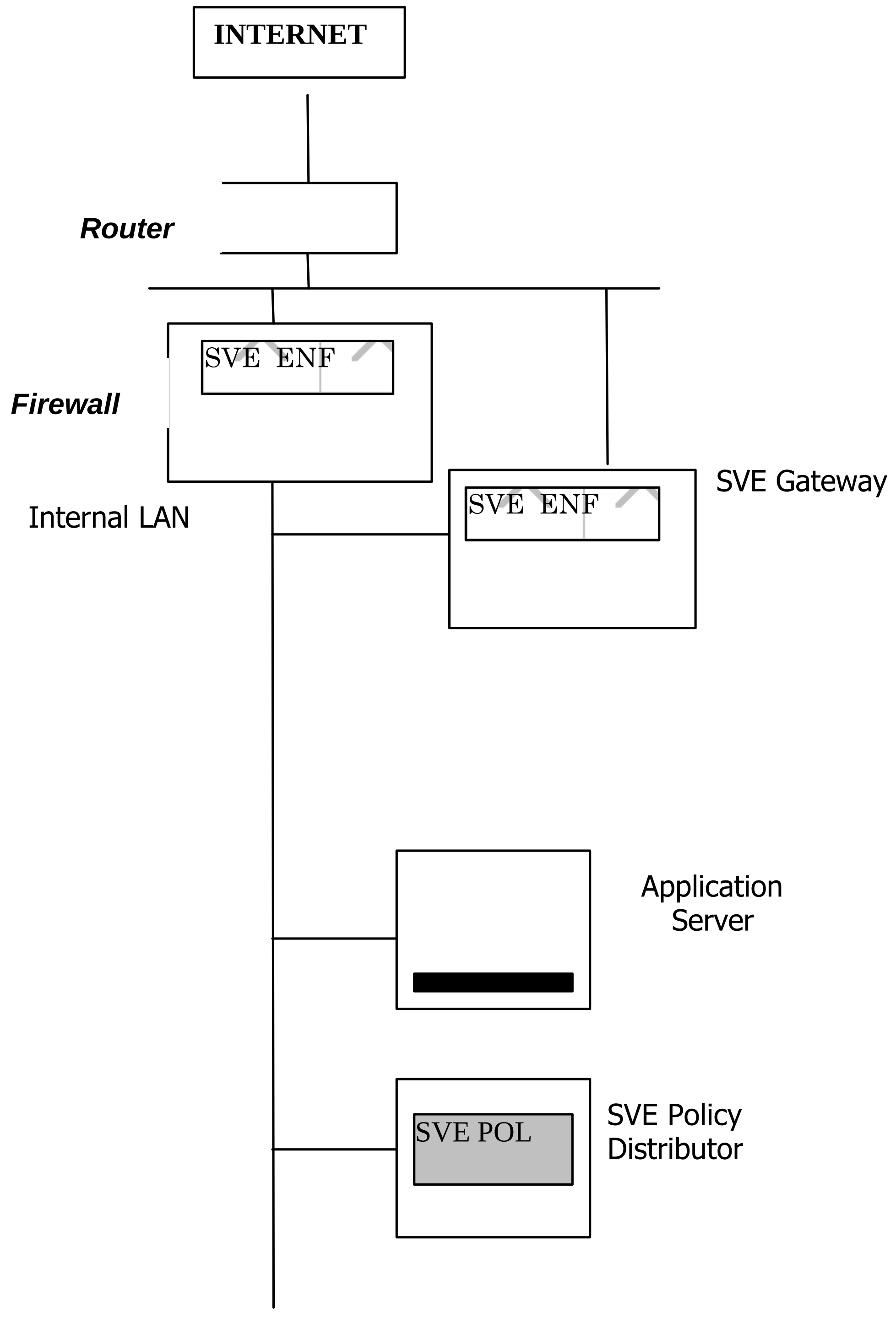

Figure 3: Multiple SVE Boundary Controllers
with Common Policy Distributor

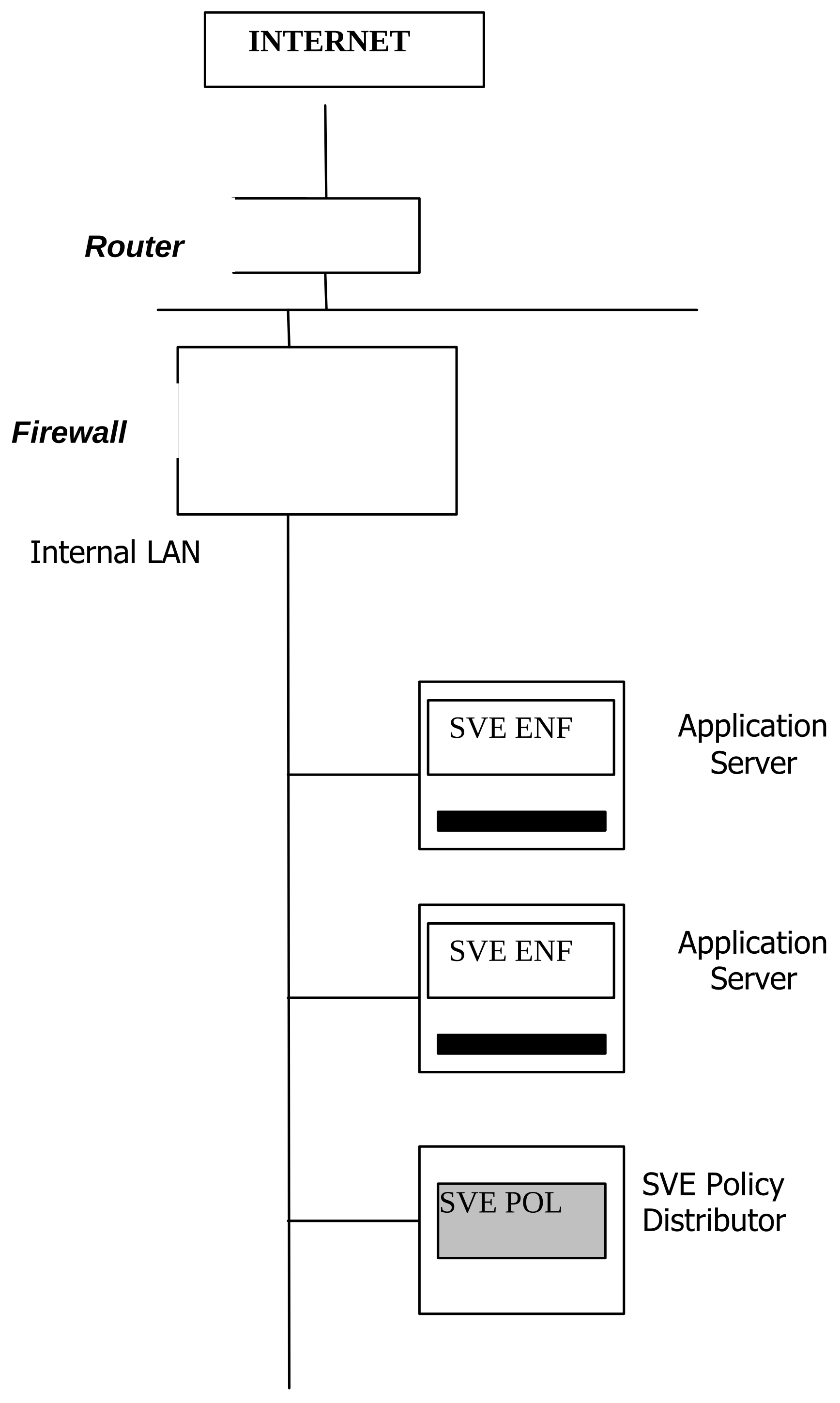

Figure 4: End-System Deployment of SVE

As in Figure 3, it is most likely that the SVE policy update distribution component (SVE POL) would reside on a different host, and distribute policy updates to multiple instances of SVE-capable end-systems. The SVE POL component would logically be a central distribution point within an enclave (and single destination for policy updates from other enclaves) but could actually be replicated for fault tolerance and scalability.

5.5 Relationship to Web Gateways

The boundary/end-system dichotomy is only absolute in cases where the inside SVE-accessible resource and the outside accessor share the same distributed application technology and the same security technology. However, application technology gateways offer a third point at which to interpose SVE security controls on inter-enclave interaction. By far the most common kind of gateway (and the one focused on by our initial investigations) a Web gateway. In Web gateway schemes, the client software is a Web browser, the communication protocol is HTTP, and some amount of application is embedded in HTML, typically forms for users to fill out to provide input parameters for application requests. The great advantage is that practically everyone has a browser, and there is no need to provision user workstations with application-specific client software. Of course, there are many advantages to the use of application-specific client software (including security benefits) but many significant distributed applications also have a Web front end as a low-impact alternative for casual users.

The control point for SVE security is on the Web Gateway Server, which may also be a full service WWW server, or may be a component dedicated to Web application gatewaying. In either case, the SVE controls are interposed at the point where the browser's HTTP request is about to be passed to a gateway program (e.g., a CGI program). SVE controls can govern which gateway programs are executable by which collaborators. Control can be coarse or fine-grained depending on the application. For some applications, there are several distinct gateway programs, each for a distinct function that can almost be considered a distinct operation or method. Web-to-DBMS connectivity products are commonly have this structure, where each gateway program implements a "canned query" or other operation on a database to be accessed via the Web. Each query or other operation can be access-controlled separately, so an SVE export could be a set of these operations that have common security requirements. Web/CORBA products have similar properties that are even better.

At the other extreme are applications in which most or all of the externally accessible application functionality is embedded in a single gateway program. In such cases, SVE security mechanisms can be applied only in a coarse-grained manner.

Figure 5 shows the deployment of SVE mechanisms in Web gateway. As in Figure 4, the firewall's role is to permit outside traffic to the Web gateway, and perhaps to protect it by ensuring that all connections to it are SSL on the expected port. The Web gateway uses SSL authentication (either client certificates or HTTP-S passwords) to determine whether the requestor is a collaborator authorized by the SVE policy to access the requested resource, i.e. execute the request gateway program. If so, then the requested gateway program is executed, and application client software on the Web Gateway host communicates over the network with application server software.

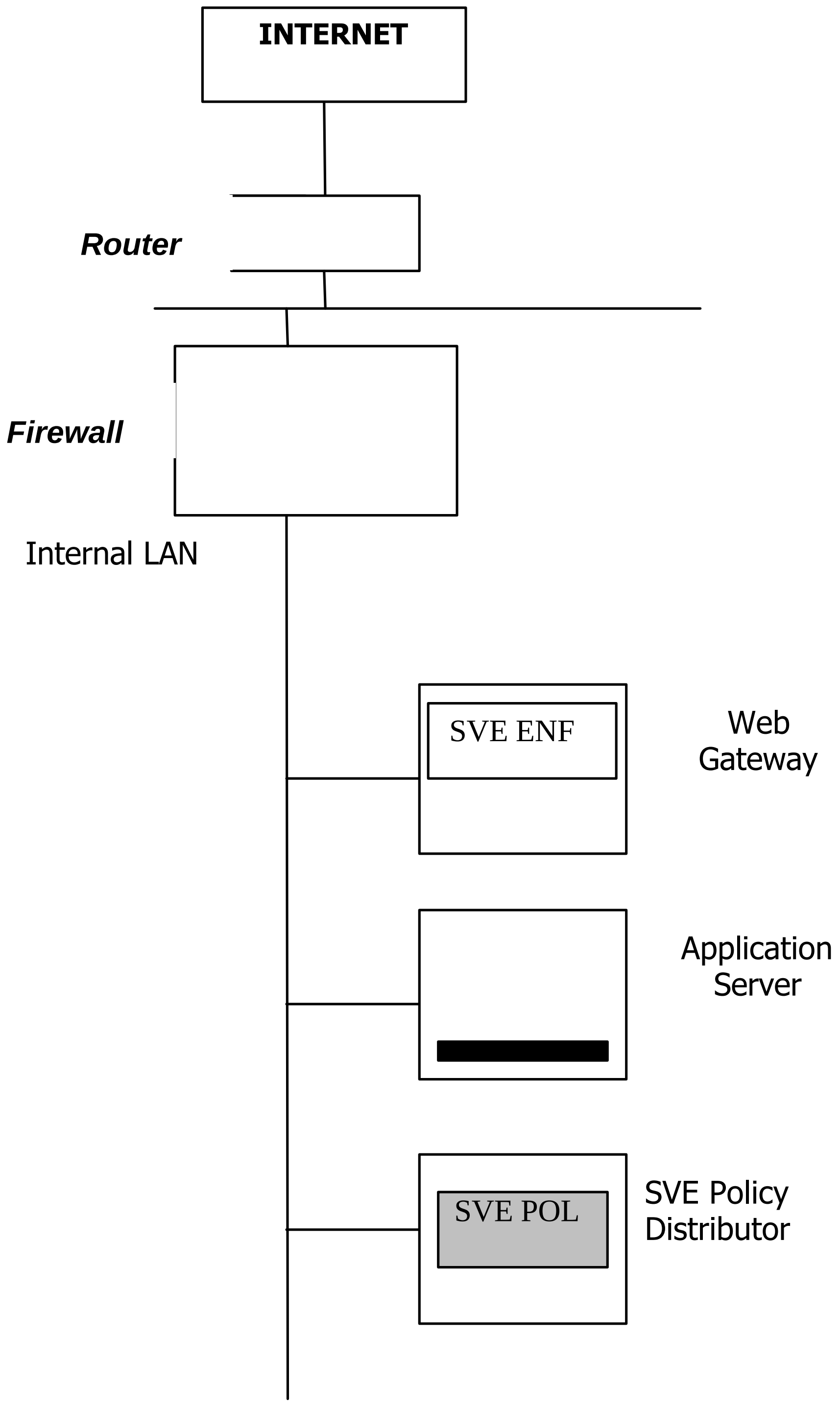

Figure 5: SVE Deployed on Web Gateway

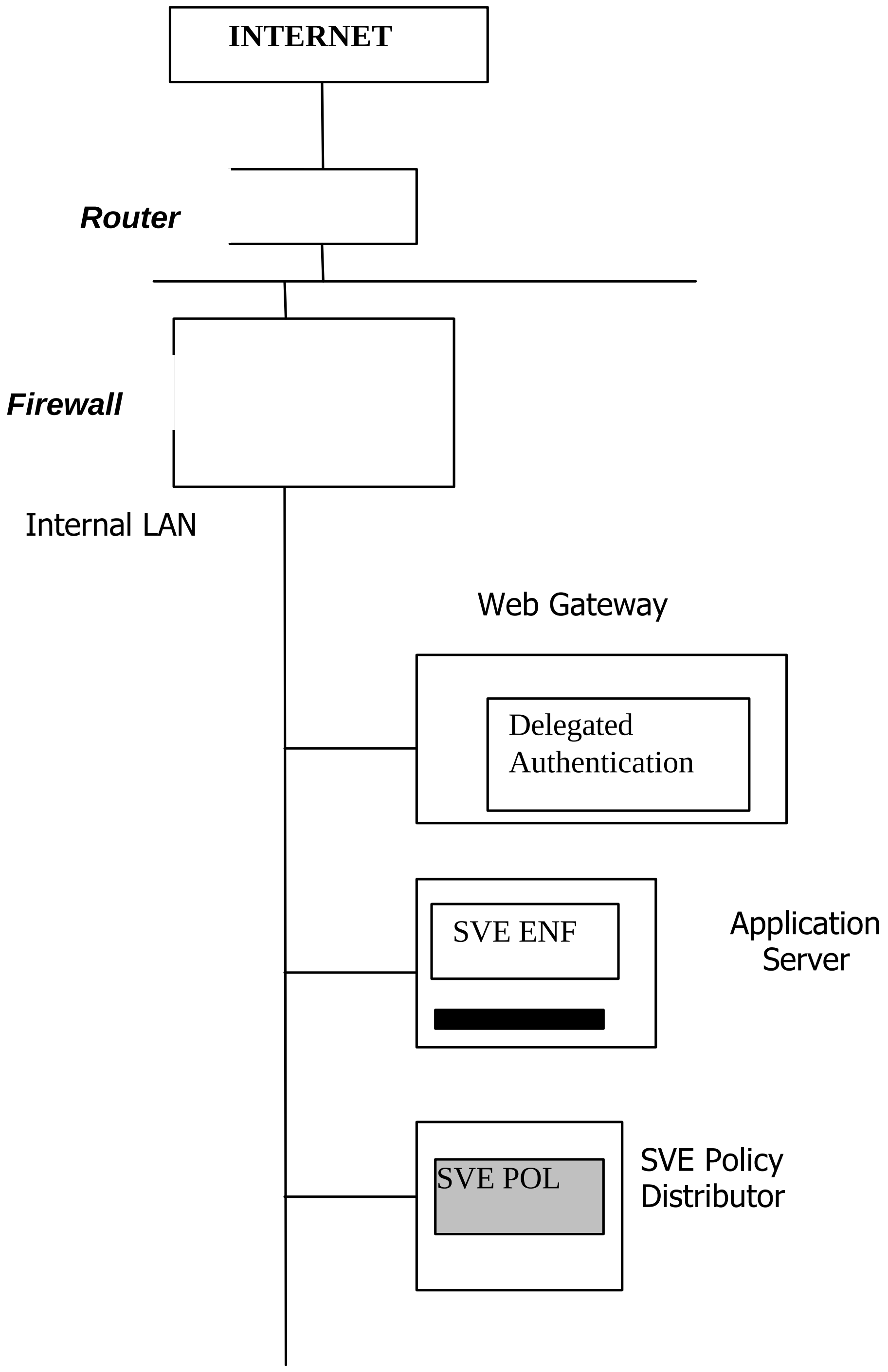

Figure 6: SVE Deployed on End System with Authentication Support from Web Gateway

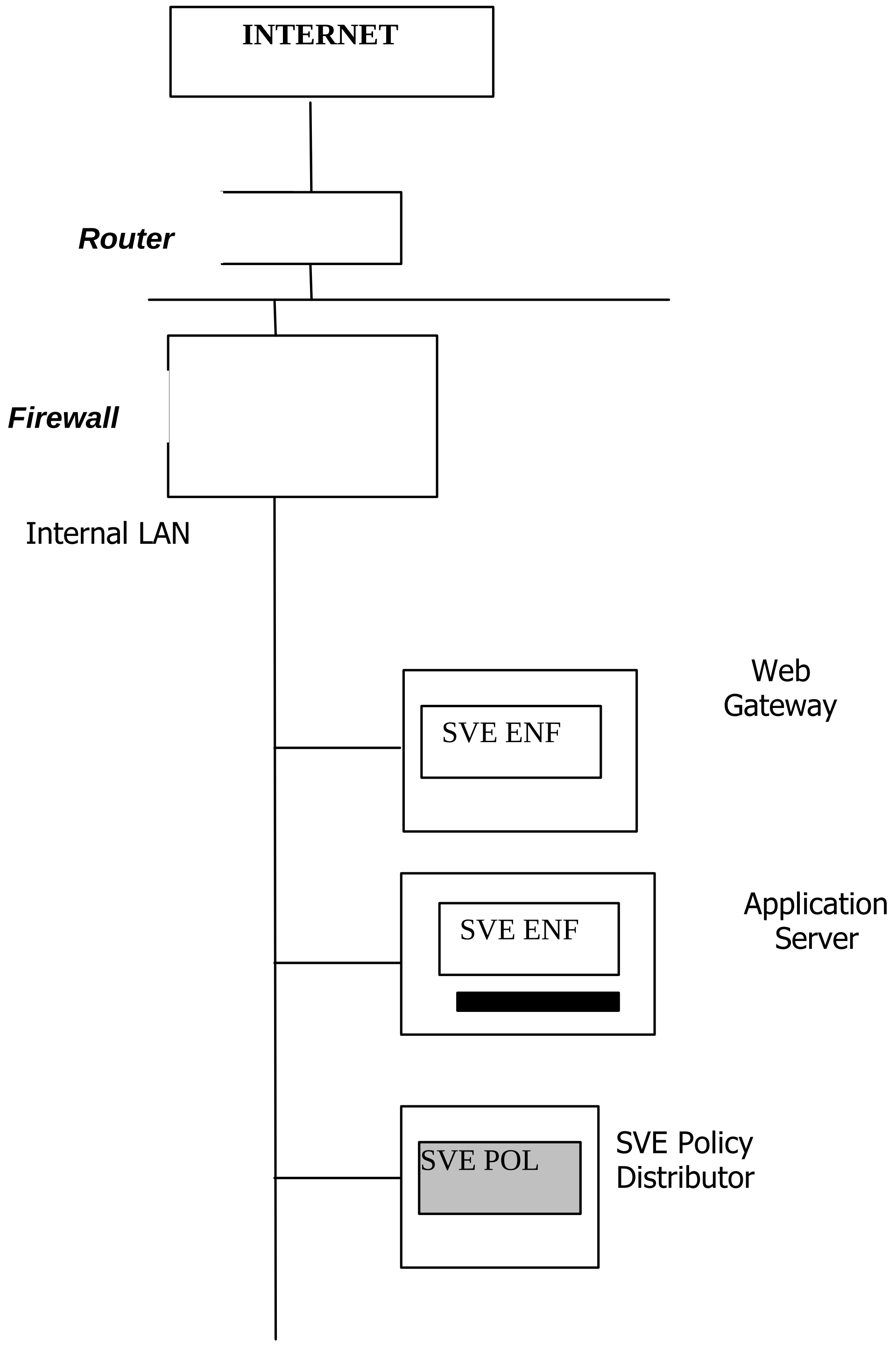

Figure 7: SVE Deployed on Web Gateway and on End System

Figure 6 shows a scheme whereby the end-system performs the SVE security checks, but relies on the Web gateway to report the authentication data of the original requestor. This approach is most applicable to distributed object technologies in which there is support in a distributed security technology for some form of simple authentication delegation.

Figure 7 shows a hybrid scheme in which the Web gateway performs the check on gateway program execution as in Figure 5, but also conveys authentication data to the end system for checks as in Figure 6.

5.6 Early Investigation and Prototyping

Thus far, we have established a number of application targets at which to deploy SVE policy enforcement components. One of these, the Web gateway, has been the focus of early prototyping because of the availability of the Java Web server. The Java Web server's native Java-based extensibility features, SSL support, and access control primitives offer significant promise of an early demonstration of insertion of SVE policy enforcement components. We are currently engaged in determining whether it is feasible to do the following:

- modify the server to support SVE (or rather to call on a stub that simulates key aspects of yet to be developed SVE policy enforcement);
- modify SSL usage to support the SVE concept of principal recognition rules;
- to do these modifications in a way that yields a more minimal, better-structured, higher assurance Web gateway than the common approach of using a low-assurance, complex full-service Web server as a gateway.

More information on the results of the investigation and prototyping plans is given in the next section.

In addition to Web gateways, we have identified CORBA as feasible target for both boundary control and end system control. This hardly surprising result is an outgrowth of work on the Sigma project, which already proved two basic concepts for SVE: the boundary-interceptor for enforcing access controls on outside access to CORBA applications (ORB Gateway); and the middleware-interceptor using similar access control technology to the boundary-interceptor. In SVE, we will demonstrate the use of the same security component (SVE policy enforcement) at both boundary and end-system (and perhaps also in a Web gateway for CORBA applications that are also Web front-ended). As a result, we will be able to demonstrate the SVE benefits of dynamic collaborative policy, multi-faceted policy (not only access control), and coordinated policy management among several SVE enforcement components deployed at various places in an enclave.

DCOM/OLE, like CORBA, is a distributed object middleware system, and should in theory be amenable to similar treatment. However, the protocols differ, the security technology is not mature or in wide use, and support for interceptors or plug-ins in unclear. DCOM will be a major focus of continuing investigation to determine how to most feasibly support this important application technology. At a fallback, we can demonstrate Web gateway front-ends to DCOM applications as one means of sharing DCOM applications between enclaves.

Java is the third distributed object application technology. Early investigation suggests the feasibility of proxying the Java RMI protocol in a manner similar the ORB Gateway's handling of IIOP. End-system support for SVE in Java servers seemed less promising, but the new Java server/servlet technology (on which the Java Web server is based) may offer some new approaches.

Finally, we do plan to demonstrate the use of SVE security in conjunction with some kind of applications that are not distributed object applications based on middleware. However, because of the greater degree of integration work for objects and middleware, we have focused our investigations on these. We are confident that at the very least some combination of firewall proxy techniques and SVE techniques (perhaps deployed as in Figure 2 along with CORBA, COM, or Java SVE gateways) can be feasibly demonstrated.

5.7 Results of Early Investigation

The Java Web Server (JWS) is currently being used as the basis of an early Web gateway prototype. The purpose in producing this rudimentary prototype was to become familiar with the JWS security model and the native mechanisms used to enforce access control on resources available through the server.

The JWS security model supports the concept of realms, resources, access control lists (ACLs), groups and users. A realm is defined as a domain space that is managed by the JWS. Contained within a given realm are resource, ACL, group and user objects that are managed by the realm. A resource object is defined as a file or gateway program that can be accessed by an http client. It may also have an associated ACL to provide an access control decision. A group is defined as an equivalence class of users, and both groups and users may be associated with any number of ACLs within a realm.

As an example, an http client that requests a protected resource will first be authenticated by the JWS as a user in the realm that contains the resource. Authentication by the JWS is currently limited to http basic or digest authentication but may be extended to include the use of client certificates or possibly some other authentication mechanism. If authentication is successful, access will be granted based on the user or group access rules defined in the ACL that is associated with the target resource.

The JWS is extensible through a standard Java Servlet Application Programmer Interface (API). Our initial investigation into the JWS suggests that resource, ACL, group and user objects for a given realm may be programatically manipulated through a custom servlet. This appears to open up the possibility of enforcing a dynamic SVE security policy through a custom servlet that maps SVE principal recognition and access control rules to native JWS security mechanisms.

The concept of using servlets for adding SVE functionality to the JWS is very attractive at this point. In addition to extensibility, the use of servlets means that we are free to explore the capabilities of other web servers (Apache, Jigsaw, Microsoft IIS all support the Java Servlet API) should the JWS not meet our expectations.

This work is sponsored by DARPA and managed by USAF Rome Laboratory under contract number F30602-97-C-0269.