A security design for a wide-area distributed system

Leiwo, J.; Haenle, C.; Homburg, P.

published in
Proc. Second International Conference Information Security and Cryptology (ICISC'99), Seoul, South Korea, December 1999. In LNCS 1787

1999

document version
Publisher's PDF, also known as Version of record

A Security Design for a Wide-Area Distributed System

Jussipekka Leiwo[1], Christoph Hänle[1], Philip Homburg[1], Chandana Gamage[2] and Andrew S. Tanenbaum[1]

[1] Vrije Universiteit, Faculty of Sciences, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, {leiwo,chris,philip,ast}@cs.vu.nl
[2] Monash University, PSCIT, McMahon's Road, Frankston, VIC 3199, Australia, chandag@pscit.monash.edu.au

Abstract. Designing security of wide-area distributed systems is a highly complicated task. The complexity of underlying distribution and replication infrastructures together with the diversity of application scenarios increases the number of security requirements that must be addressed. High assurance requires the security enforcement to be isolated from non-security relevant functions and limited in the size of implementation. The major challenge in the is to find a balance between the diversity of security requirements and the need for high assurance. This paper addresses this conflict using Globe system as a reference framework, and establishes a security design that provides a flexible means of addressing the variety of security requirements of different application domains.

1 Introduction

Security design refers to the interfaces and services that must be incorporated into the system to enable addressing of different security requirements [5]. The security design must be such that it enables verification and validation of the security enforcement to achieve high assurance. Assurance refers to the confidence that the security enforcement is appropriate.

A number of generic considerations must be addressed by the security design to achieve high assurance. For example, the amount of trusted code should be kept to a minimum, duplicate security functions should be eliminated, the trusted code should be designed to enable verification and validation, and the software should be designed to enable code optimization for different hardware platforms [1].

Addressing these considerations leads towards a trusted computing base capable of verifiably enforcing a small number of security requirements. Diversity of security requirements is often a prohibitive factor for high assurance [8]. However, security requirements of distributed object systems are of a very high diversity. Not only secure communication requirements, e.g. encryption and authentication, but also requirements of secure operating systems, e.g. secure method or function execution and access control, and those of secure communication systems, e.g. traffic filtering and client behavior monitoring for intrusion and misuse detection, must be addressed.

Designing security on development platforms for distributed shared objects (DSO), such as Globe [15], that provide transparent distribution and replication of objects over multiple physical locations further complicates the security design. The major advantage of Globe type of systems over, say, CORBA is the scalability. CORBA, DCOM and other existing distributed object technologies assume a static replication model, whereas Globe enables per-object replication strategies allowing an increased flexibility in designing global distributed object systems [3].

Security design must also be established in a manner that allows object-specific security policies being established and maintained without limiting the range of applications [16].

A number of security architectures, such as Kerberos, Sesame, and DSSE, have been established for distributed and networked systems [9, 12]. Such architectures are mostly concerned with the development of secure applications on networked and distributed environments. They do not address the security of the inherently complicated distribution infrastructure itself. Scalability to global systems is also questionable.

This paper examines the problems associated with security designs of global DSO architectures in general, and Globe in particular. We begin by identifying the security requirements of distributed object systems and proceed by examining the challenges that the security designer faces when attempting to address these requirements. This is followed by a comparison of two possible security designs. The implementation aspects of the chosen design shall then be discussed. Finally, conclusions shall be drawn and directions highlighted for future work.

2 Security Requirements in Distributed Object Systems

Security requirements of distributed systems must be addressed through communication security, operating system security, and network security requirements. There are also object life-time requirements dealing with creation, binding and disposal of objects. Security management must be addressed, as well as educational and operational security requirements, and other pervasive security requirements.

In the following, examples of security requirements are given at each category. The list is comprehensive, yet it is questionable whether it can ever be complete. Also, the identified requirements are partially overlapping and each one may contribute to more than one security objective.

For example, message semantics based filtering of protocol messages is a means of achieving access control. If the traffic filtering is enforced at the application level, there is a close relationship to the access control based on method execution request. This is to be addressed at the operating system level. However, in most cases the traffic filtering is applied at lower levels, for example at the network layer, when it must be considered as a means of network security instead of a means of operating systems security. Both ways ways still contribute towards access control

2.1 Communication Security Requirements

Communication security is mostly concerned with cryptographic techniques to achieve confidentiality, integrity, authenticity, and non-repudiation of communicated messages. The communication security requirements of distributed object systems are not fundamentally different from general communication security requirements, for example those of the ISO OSI standard [9].

Depending on the type of communication, confidentiality can be addressed through **Connection-oriented confidentiality requirement** or **Connectionless confidentiality requirement**. Not all the protocol fields may require equal security, and it may become appropriate to address the **Selective field confidentiality requirement**. In some environment, even the fact that communication between certain hosts occurs may be sensitive, and the **Traffic flow confidentiality requirement** must be addressed.

Integrity measures may be implemented with or without recovery. Those supporting recovery allow the reconstruction of the message from the integrity check, whereas those without recovery can only be used for verifying the correctness of pairs of messages and integrity checks. As integrity can be addressed by entire protocol messages or selective fields, **Message integrity requirement with recovery**, **Message integrity requirement without recovery**, **Selective field integrity requirement with recovery**, and **Selective field integrity requirement without recovery** must be addressed.

Authentication can be applied either to a peer object or to the origin of data. Additional measures can be provided for client and user authentication. However, they should not be addressed at the technical infrastructure of object distribution. **Peer object authentication requirement** must be addressed when data from a communicating software module, such as a protocol stack implementation, must be authenticated to the peer object in communications. Closely related is the **Peer object integrity requirement** where assurance must be provided to a peer object in a communicating system of the peer object implementation not being altered by, for example, a trojan horse. **Data origin authentication requirement** addresses the authentication of the communicating hosts as sources of protocol messages.

Non-repudiation of origin requirement addresses concerns of a sender of a message repudiation participation in communication. **Non-repudiation of receipt requirement** provides means to verify that a particular recipient has in fact received a message. **Non-repudiation of delivery requirement** is more complicated. Usually, it can not be addressed at the applications level.

2.2 Operating Systems Security Requirements

Traditionally, operating system security has been concerned with access control to protect files from unauthorized reading or modification, or to prevent unauthorized execution of system administration software. In distributed object systems, object can refer to objects of any granularity. Therefore, **Object integrity requirement** and **Object confidentiality requirement** can refer to

any operating system object, or to the distributed shared object as a whole. **Object level access control requirement** refers to the measures to determine which accesses are allowed within the DSO.

From the DSO point of view, object confidentiality, integrity, and access control requirements refer to the state of the entire distributed shared object remains confidential or unaltered. Access control requirements deal with which clients are allowed to bind to the DSO. At such a coarse granularity, security issues must be addressed through object life-time security measures, addressing them at the operating system level is hard.

With the emerge of new networking technologies and new programming languages, the scope of operating systems security extends into more active control of, for example, program execution. A typical example of extended operating system security functionality is the Java Virtual Machine (JVM). Such requirements can be addressed through **Secure method execution requirements**.

In addition to addressing the communication security requirements to prevent method invocations from remote hosts being tampered with, the **Method integrity requirement** must be addressed to assure remote clients with the correctness of methods stored and executed in remote hosts. This is different from the peer object integrity requirement in a sense that it addresses the actual methods provided by the DSO, not the integrity of the methods of replication and distribution infrastructure.

2.3 Network Security Requirements

Network security is addressed to provide assurance of the correct operating of the distributed system that employs the above security technologies. The firewall capacities are provided by the **Message semantics filtering requirement** that addresses the selective transmission of protocol messages to block unwanted protocol messages from being processed by the distributed object.

At the distributed object level this means selective processing of control messages and remote method invocations. At the underlying communication infrastructure, this means selective forwarding of suspicious datagrams.

Client behavior monitoring requirement and **Method execution sequence monitoring** are means to detect intrusions and misuse and trigger appropriate alarms.

2.4 Object Life-Time Security Requirements

There are a number of security requirements that must be addressed through the object life-time instead of during the operation of the object. Most importantly, secure binding and secure disposal of the components of a DSO. When a new client wishes to connect to a DSO, it must initiate the binding procedure.

The first step in binding is that the candidate client contacts a name server to request the unique object identity that matches the symbolic name of a DSO where the connection is to be established. The name service must be protected by addressing the **secure name service** requirements.

Once in the possession of the unique object identity, the new client proceeds with the binding by contacting the location server. The location server returns the contact address of the DSO as a pair of network address and port to connect to. This phase must be protected by addressing the **secure location service** requirements.

In the following step, the new client contacts the implementation repository where the program code is loaded to construct the local representative of the DSO in a local address space. This step must be protected by the **secure implementation repository** requirement. In fact, there are a number of open issues in the security of downloading executable content that must also be addressed in case of non-local implementation repositories.

With the implementation of the local representative, the client can proceed with the binding to the DSO using the newly created local representative and the contact address received from the location service. **Secure connection establishment** requirement must address the issues related to the establishment of the connection between the local representative and the DSO. This is also the phase where end-user security requirements, such as **user authentication** must be addressed.

At the end of the life time, the local representative disconnects from the DSO. This includes disposal of the local code as well as the state of the DSO and the security state of the communication. This must be addressed through the **secure object disposal** requirement.

2.5 Pervasive Security Requirements

Pervasive security requirements can be studied from a number of points of view. One point of view is the security requirements related to the management of information security. These requirements include, for example, security planning and security maintenance. Also, a number of security education and awareness requirements must be addressed. However, as these bear no significance to the establishment of a security design for DSO systems, they shall not be further studied herein.

Other pervasive security requirements are those that are needed for properly implementing specific security requirements. For example, security labels, trusted implementation of security functions and so further must be addressed once implementing the security design in a particular application scenario.

Finally, there are a number of implementation requirements that have security relevance, even though they are not addressed through actual security measures. A typical example is the total ordering of method invocations to prevent inconsistent states of a DSO by incorrect method execution sequences. A comprehensive treatment is provided, for example, by Birman [4].

These are often considered to be issues addressed by the reliability engineering and dependable computing point of view. They are more concerned with continuity and correctness of services in the presence of random and independent failures, not in the presence of active attacks, i.e. selective failures. Therefore, they shall not be made a part of the security design.

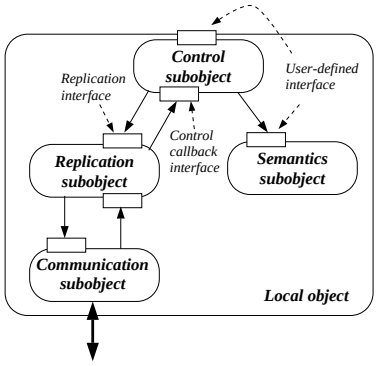

Fig. 1. Implementation of a Globe local object

3 Challenges of Security Design

To examine the challenges of security design in wide area distributed systems, the Globe object architecture will be used in this paper as a concrete example. After the introduction of the Globe object architecture, three security design challenges shall be addressed: selection of the security design model, coping with the diversity of security requirements, and the placement of security measures within the object architecture.

3.1 Globe Object Architecture

A central construct in the Globe architecture[3] is a distributed object. A distributed object is built from a number of local objects that reside in a single address space and communicate with local objects residing in other address spaces.

Together, the local objects form the implementation of a particular DSO. Local objects consist of the actual interface and the distribution mechanism, as illustrated in Fig. 1. The distribution mechanism enables transparent distribution and replication of objects, hiding details from application developers.

The semantics subobject contains the methods for the functionality of the DSO. This is the only subobject the application developer must develop himself. It is as objects in middleware architectures such as DCOM and CORBA.

The communication subobject is responsible for the communication between local objects residing at different address spaces. It implements a standard interface but can have several implementations depending on the particular communication needs and provides a platform-independent abstraction of underlying networks and operating systems providing the communication services.

The replication subobject replicates and caches the local objects and constructs the DSO from local objects. It also implements coherence protocols to

[3] More details available at http://www.cs.vu.nl/globe/

decide when methods of the local semantics subobject can be invoked without violating the consistency policy.

The control subobject invokes the semantics subobject's methods. It also marshalls and unmarshalls invocation requests passed between itself and the replication subobject.

3.2 Security Design Model

Several security design models have been established over years. Most of them, however, focus on multilevel secure systems and databases (e.g. [1, 6, 14]) instead on the security of conventional applications on general purpose operating systems. High dependence on risk analysis further limits the applicability of many models (e.g. [2, 13]).

Limitations of risk analysis become evident in the design of the security of system development and distribution platforms. Since underlying implementation technologies and operational environment are not known at the time of security design, neither threats nor losses can be estimated. However, a systematic approach, such as [10], is required for guiding the security design.

The major advantages of [10] is that it reduces risk analysis into a decision making tool and is heavily based on the Common Criteria for security evaluation [7]. It divides security development into three stages. First one deals with the specification of all relevant security functions capable of satisfying a certain security objective. The second stage aims at selecting a subset of all possible security requirements to be implemented in a particular system. Selected measures are implemented and evaluated at the third stage.

This paper deals with the first stage of the model. A possible set of security requirements of a distributed object system are identified and a security design is established to aid in the implementation of security measures to address those requirements. When designing applications using Globe, the particular operating system and communication mechanisms can be selected, and a subset of possible security countermeasures implemented to match the specific application level security policy.

3.3 Diversity of Security Requirements

The ideal case of security in distributed object systems is a dedicated security subobject that implements security measures similarly to a traditional reference monitor. However, it is not obvious how this can be achieved in practice taking into account the high diversity of security requirements and the possibility of security requirement being addressed at different subobjects. The possible components where different security requirements of distributed object systems can be addressed within the Globe object architecture are illustrated in Table 1.

The semantics subobject does not have security relevance to the Globe architecture since it does not participate in replication and distribution. However, a high number of security requirements may be addressed at the application level through the semantics subobject.

Table 1. Possible security requirements of Globe as divided to the underlying communication infrastructure (UCI), communication subobject (CoS), replication subobject (RS), control subobject (CS), application (AL), and operating system (OS)

Requirement	*UCI*	*CoS*	*RS*	*CS*	*AL*	*OS*
Connection-oriented confidentiality requirement	X	X	X	X	X	X
Connectionless confidentiality requirement	X	X	X	X	X	X
Selective field confidentiality requirement	X	X	X	X	X	X
Traffic flow confidentiality requirement	X	X	X	X	X	X
Message integrity requirement with recovery	X	X	X	X	X	X
Message integrity requirement without recovery	X	X	X	X	X	X
Selective field integrity requirement with recovery	X	X	X	X	X	X
Selective field integrity requirement without recovery	X	X	X	X	X	X
Peer-object integrity requirement	X	X	X	X	X	X
Message semantics filtering requirement	X	X	X	X	X	X
Peer entity authentication requirement	X	X	X	X	X	X
Data origin authentication requirement	X	X	X	X	X	X
Non-repudiation of origin requirement	X	X	X	X	X	X
Non-repudiation of receipt requirement	X	X	X	X	X	X
Non-repudiation of delivery requirement	X					
Object level access control requirement			X	X	X	X
Client behavior monitoring requirement				X	X	X
Method execution sequence monitoring requirement				X	X	X
Method integrity requirement			X		X	X
Object integrity requirement			X		X	X
Object confidentiality requirement			X		X	X
Secure method execution requirement				X		X

The security architecture is also independent of the underlying communication architecture. A TCP/IP network could implement packet confidentiality and authenticity in form of IP SEC standard, or a transport layer security by SSL or SSH. In more advanced scenarios, network traffic could be authenticated and access control provided by Kerberos, Sesame, or DSSA/SPX. Since the objective of Globe is flexibility and platform independence, no assumptions of the available security services can be made. All communication security measures may need to be implemented at the communication subobject.

The communication subobject security is concerned with secure communication channels between local objects. Requirements are those of secure communication, i.e. confidentiality, integrity, authenticity, and non repudiation. Access control can be enforced through traffic filtering based on protocol messages.

In group communication, communication security measures can be applied at the replication subobject or control subobject on per-message rather than per-recipient basis. If the communication subobject manages group communication through a number of point-to-point channels, this can significantly reduce the cryptographic overhead.

The replication subobject security is also concerned with the enforcement of

secure replication of objects, and prevention of malicious parties from altering the DSO state, interface or implementation.

The control subobject and the underlying operating system are responsible of secure execution of the methods of the semantics subobject. Method level access control can be provided to decide which methods can, under which constraints, be invoked in the local environment. Client behavior and method execution sequences can be monitored for intrusion and misuse detection.

3.4 Placement of Security Functionality

Consider a simple entity authentication protocol [11, p.402]. B initiates the protocol by sending a random value r_B to A. A replies with a random number r_A and a keyed hash $h_K(r_A, r_B, B)$. B then sends the value $h_K(r_B, r_A, A)$ to A allowing both parties to verify each other's authenticity through knowledge of key K shared by A and B.

$$A \leftarrow B : r_B \tag{1}$$
$$A \rightarrow B : r_A, h_K(r_A, r_B, B) \tag{2}$$
$$A \leftarrow B : h_K(r_B, r_A, A) \tag{3}$$

This (or similar) protocol is likely to be implemented at several subobjects requiring peer-object authentication. An obvious design is to separate calculation of the hash value from the protocol execution logic. Protocol implementation becomes easier as the hash function can be treated as a black box and implemented separately, possibly in hardware.

Availability of a mutually agreed upon hash-function between entities is required in steps (2) and (3). To negotiate the function and to store security state, such as cryptographic keys, a security context must be maintained by the communication parties.

The protocol logic can be implemented as a separate function or as a, say, Java object (not a Globe object, though), that is called or instantiated by objects requiring to authenticate with their peer-objects. The subobject instantiates the authentication object and defines the parameters, such as the behavior in error conditions. This complicates the object interface but provides high encapsulation of security relevant processing in a dedicated authentication protocol object.

Not all protocol errors imply an authentication failure. They may be due to network congestion, excessive workload at the peer object, or some other random condition occuring. The recovery logic must decide which action to take, whether to deal with the peer-object as un-authentic, to proceed with the service request and assume further authentication at other subobjects, to block the service request and retry authentication after a delay, or to take some other action.

Parameterization of all possible error conditions at different subobjects leads to a complex exception handler or to an increased interaction between the security subobject and conventional subobjects, This reduces functional cohesion of the security subobject and leads to weaker encapsulation.

An attempt to isolate the security functionality has, therefore, led to an increased complexity of the protocol object. Each authentication request must be related to a number of other security-relevant objects, such as security context and exception handler. Proper software engineering practice, such as thin interfaces and functional encapsulation can, however, improve the design.

Each subobject has to be given a unique identity, expressed as the identity of a local object and the particular subobject. This identity is used for authentication. The protocol execution logic can be easily separated from the subobject but the semantics of different protocol messages has to be bound to a particular subobject. This encourages implementation of the authentication protocol as part of the conventional subobject as it is mostly aware of various subobject-specific semantic conventions.

The denial of service aspects should also be kept in mind. Globe objects are typically distributed using public networks, e.g. the Internet, with limited quality of service guarantees. For example, TCP guarantees an ordered delivery of messages but not the maximum time for message transmission. Protocols that depend on a TCP connection between two hosts may cause serious performance penalties due to network congestion outside the control of any local object.

Resource allocation policies may be defined for protocol steps or stateless protocols designed to prevent denial of service. However, protocols have to be carefully designed and evaluated for optimal performance and reaction on exceptional conditions. Therefore, they should be dealt with as independent software artifacts.

Protocol implementations are also likely to require a preparedness plan in terms of exception handling to recover from situations where a critical resource, such as protocol execution time, exceeds a threshold. Recovery is very subobject specific and difficult to generalize into a common protocol implementation.

These issues complicate the decision of whether the security should be managed by a dedicated security subobject, or by each conventional subobject independently. The following section provides a detailed analysis and evaluation of the two alternatives.

4 The Security Design

A generic security design for a DSO, as illustrated in Fig. 2, consists of a security subobject, security policy and a number of security associations (SA). SAs describe the security state of communication channels and may be shared by multiple parties. They contain, at least, encryption and authentication keys, modes of algorithms, and other parameters such as initialization vectors, and the SA life time.

Prior to secure communication, peer objects must establish a SA through on line or off line negotiations. The initial state of the security association is downloaded during the binding. The number of security associations maintained by a local object may be different in different implementations and application environments.

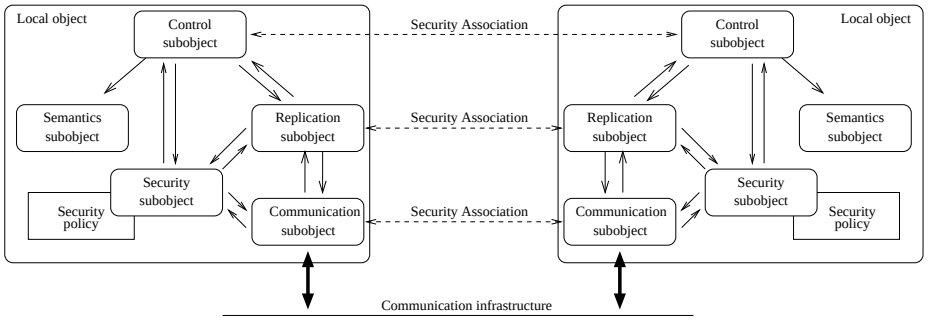

Fig. 2. A Generic DSO security design

The SA must be supported by a security subobject that contains the implementation of corresponding security measures. Security subobject implements a certain communication security policy. Communication security policy is fairly static but security associations may change dynamically. Additional security policies are required for access control and intrusion detection.

Replacing a communication security policy means binding to a different security subobject. Full policy-mechanism –independence, as in access control models, is hard to achieve due to the difficulties of formally expressing communication security requirements. The local object may initially contain an implementation of a number of security subobjects or download them when necessary.

There are two main alternatives for coordinating the security enforcement:

Centralized security coordination (CSC) where conventional subobjects request all security measures to be executed by the security subobject.

Distributed security coordination (DSC) where conventional subobjects execute the security measure but depend on the security subobject for critical functions, such as encryption and decryption of buffers.

In CSC, the communication security policy is followed by the security subobject that enforces the policy and executes the required security measures. In DSC, the communication security policy is followed by conventional subobjects.

In the following, the two shall be compared and evaluated against a number of security design criteria. The comparison is followed by a discussion about the security association and different security policies. A clear distinction between the two is impossible in practical systems. The implemented system is likely to be a hybrid. However, the comparison suggests that implemented systems should bear more characteristics of centralized than distributed security coordination.

4.1 Centralized Security Coordination

Communication between local objects is mediated by the security subobject. The security subobject maintains the security associations and negotiates their

content with the security subobject of the peer local object. Certain security processing of messages is carried out at each passing of messages between different subobjects.

Prior to passing a protocol message to a lower or upper level in the subobject hierarchy, a subobject passes it to the security subobject. The security subobject applies the security measures to protocol messages and returns. The subobject that called the security subobject passes the security enhanced protocol message to the next subobject. The same process is repeated at each subobject and reversed at the receiving local object.

As a minimum, the security subobject only requires an interface for passing and receiving messages to and from subobjects. Each subobject interfaces with the security subobject through a common interface that may have different implementations. The calling sequence from subobjects to the security subobject can be standardized. This allows replacement of the security subobject without modifying other subobjects.

Full isolation of the security subobject is hard to achieve, mostly due to exception handling. Which action is taken if a security measure, e.g. data origin authentication, fails? The interface can be standardized to return a number of status codes the conventional subobject can use for examining the status of the security processing. For example, a standardized security exception could be thrown by the security subobject and caught by the conventional subobject. Implementations on languages such as C may return a standardized error code.

4.2 Distributed Security Coordination

The security subobject only provides basic security mechanisms to aid subobjects in the enforcement of security as part of the subobject functionality. The conventional subobject maintains the SAs and implements the security logic.

The security subobject provides basic security services, such as encryption and decryption of buffers, generation and verification of authentication codes of buffers and so on. The conventional subobject will call these measure when necessary according to the security logic. The receiving conventional subobject implements corresponding security protocols, and executes the actions of a receiving entity of the protocol.

This approach introduces a classical trade–off between assurance and diversity of security requirements. High assurance can be provided for, say, cryptographic processing on this scheme with the cost of reducing the security subobject functionality and complicating the security implementation. In CSC, equally high assurance can be achieved on cryptographic modules, but in general, higher assurance can be achieved to the general security processing.

The complexity can be reduced by standardized security libraries and proper software engineering practices. Replacing and extending the security subobject remains easy. Exception handling is logically connected to the protocol execution. Adapting to a different communication security policy remains complicated, though, alterations are required to each conventional subobject.

Table 2. Comparison of security subobject designs

Criteria	*CSC design*	*DSC design*
Economy of mechanism	Average	Average
Duplicate functions	Good	Poor
Optimization for new hardware	Good	Good
Complete mediation	Good	Poor
Least privilege	Good	Poor
Future alterations	Good	Average
Ease of SA maintenance	Good	Poor

4.3 Comparison of Designs

Many principles of security design (e.g. [1, 14]) focus on access control models and their applications. In the following, the above design alternatives are evaluated against security design principles adapted to the DSO context. Findings are summarized in Table 2.

Economy of mechanism refers to the small size of implementation and simplicity of the design to enable appropriate testing and evaluation. It is unlikely that either design can achieve such economy of mechanisms to enable formal verification of security. Yet, this would be unnecessary in most application scenarios, due to general purpose operating systems used. There are no significant differences between the two designs.

Elimination of duplicate functionality requires that no security functionality should be implemented in multiple modules. This is a significant disadvantage of the DSC design. Many of subobject's security protocols are likely to be similar and must be implemented by each subobject even though most central security functions are implemented by the security subobject. In the CSC design, each protocol is implemented once. This improves the control of the implementation but increases the complexity of security subobject's interface.

Code optimization for new hardware is essential for performance reasons. It is likely that only cryptographic functions are implemented in hardware. In both designs, the cryptographic functions can be separated from protocols by proper software engineering or by logical separation of security protocols from security functions. Both designs provide a considerably good support for hardware implementations of different security functions.

Complete mediation requires the security subobject being consulted in each method invocation. System design should prevent subobjects from bypassing security on discretion. Complete mediation can, through design, be achieved by the CSC design. The security subobject methods are always called, even though no security measures are applied (i.e. some security functions are NULLs). With DSC, control measures and security method integrity checks have to be applied at multiple locations.

Least privilege refers to the components of the system gaining only a minimum set of accesses to sensitive data required for completing their tasks. The DSC design is problematic because of the distribution of security related pro-

cessing to every subobject of the system. Each subobject must be given access to security critical components, such as security associations. The CSC design enables easier control of the privileges.

Ease of future alterations measures adaptation to different security policies. As security requires continuous maintenance, this is an important criteria. The CSC design is superior, mostly due to the single point of alterations required. In DSC design, each alteration in protocols of security functions must be implemented in each subobject. However, proper software engineering can simplify alterations.

The comparison suggests the superiority of the CSC design over the DSC design. However, the comparison only deals with the design criteria, not on performance issues. It is not clear how significant performance reduction is caused by security processing relative to, for example, network latencies when the local objects are distributed over wide area networks. Intuitively, it appears that performance penalties of different designs are not significantly different relative to the overall cost of communication. Real measurements are required to confirm the intuition.

The CSC design as has some additional advantages over the conventional subobject enforced security in, for example, **ease of SA maintenance** as discussed in the following section.

4.4 Security Association

Each local object must share at least one SA with local objects it is communicating with. In the CSC design, the security associations are maintained by the security subobject.

SA does not have any particular functionality. It is a data structure that stores the security state of communication. A significant advantage of CSC design over the DSC design is in the ease of applying different SA schemes, for example

Single local object SA scheme is where a single SA is maintained by local objects and used for all security needs.

Multiple local object SAs scheme is where a number of SAs are maintained by local objects and used for different security needs but shared between all subobjects.

Single subobject SA scheme is where each subobject of a local object maintains a SA with peer subobjects and use it for all their security needs.

Multiple subobject SAs scheme is where each subobject of a local object maintains a number of SAs with peer subobjects and uses them for different security needs.

In many point-to-point applications, the single local object SA scheme is the most likely scenario. Since the subobjects of a local object are maintained in a single address space, there are no reliable means from preventing malicious local objects from violating the security of other subobjects. Therefore, multiple SAs may not be meaningful.

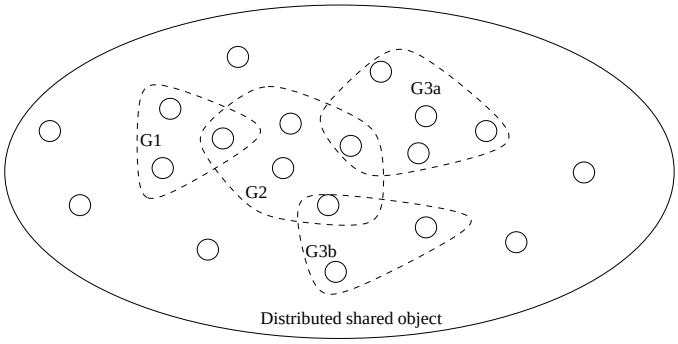

Fig. 3. A Large DSO with multiple groups of local objects

Different keys may be maintained for different services or security levels but shared between each subobject of a local object. If local objects operate on environments that provide separate address spaces or other tamper-proof execution environment, more complicated and fine-grained keying schemes may be relevant.

Complicated SA schemes occur also in very large DSOs (Fig. 3). Circles illustrate local objects that constitute the distributed object. Local objects are further grouped into three.

The core group ($G1$) is formulated of those local objects that are most crucial to the application, for example sites from which a WWW page can be updated. The cache group ($G2$) is a set of passive sites replicating the service, in this example the pages, without altering the content. The client groups ($G3a$ and $G3b$) contain those clients that have at a certain point of time registered with a member of the cache group to access the service. Disconnected local objects may connect to the distributed shared object in the future.

Assume that core group members deliver on line a data item, such as a newspaper, software component, or a digital media clip, for which a payment is required. In global distribution, the core group objects can not deliver the item to millions of customers. Rather a number of caching sites are established and clients access cache sites for the service.

The data item can be protected by encryption and registered clients can obtain (maybe once a payment transaction is completed) a cryptographic key to recover the item. Key distribution may depend on the level of trust of cache group members:

Untrusted cache is where the core group members do not trust members of the cache group. Caches hold encrypted data but can not decrypt it. Clients must buy the decryption key directly from core group members or a dedicated key server.

Trusted cache is where the core group members allow cache group members to have the encryption key of data elements. This means, data can be stored

Table 3. Security subobject enforceable security policies

Policy	*Purpose*
Communication security policy	Static Communication security
Security Association	Dynamic communication security
Access control policies	Method invocation control
Behavior monitoring policies	Intrusion and misuse detection
Local policies	For subobject internal security

in plaintext on the caches and link encrypted when communicating with a client.

The level of trust of replicas depends on the application domain. Through appropriate implementations, the need for a number of SAs can be also reduced. Untrusted caches can, for example, share one SA with a core group member and another SA with a client group member. With tamper-proof hardware, data can be decrypted and reencrypted without disclosure to the caching site.

As this may be a practical impossibility, the need for flexible SA schemes remains. The above listed SA schemes can be extended by various group-security SA schemes and application specific schemes. The security subobject can independently maintain the required SA scheme without violating the security model.

4.5 Security Policy

The local object design requires a number of security policies as introduced in Table 3. System level and managerial security policies are omitted as they are beyond the scope of Globe security design.

The largely static communication security policy that describes the security measures to be applied in the communication is implicit. It is constituted by the implementation of the security subobject. Means to achieve higher policy-mechanism independence are a major area of future research.

Lower level security policy describing the ways in which the implemented security measures are executed is expressed in a more flexible manner through dynamically changing security associations. Negotiation mechanisms of security associations in fact are mechanisms for negotiating the security policy and the components of a SA used for enforcing the policy.

Access control policies describe which methods can be invoked by which clients under which circumstances. Behavior monitoring policies describe the ways in which method invocations are monitored for intrusion and misuse detection, and how deviations are handled.

Local policies must be enforced by each subobject internally. The are not concerned with the security of communication but the internal security of a subobject. For example, prevention of denial of service attacks may require local resource allocation policies at each subobject.

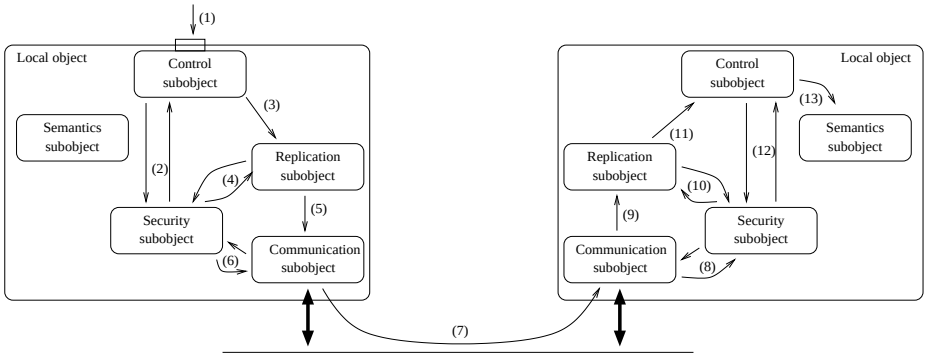

Fig. 4. Security subobject calling sequence

5 Interfaces to the Security Subobject

The DSO security has two distinctive facets: transformational security to enforce the communication security, and access control to impose restrictions on method invocations. Interfaces of the security subobject shall be discussed from both points of view. Secure operations of a DSO shall then be addressed.

5.1 Transformational Security

The security subobject must have a standardized interface and a calling sequence from other subobjects. This enables on line adoption of different communication security policies by replacing the security subobjects of relevant local objects. No alterations to other subobjects are required.

Methods of the security subobject must be paired so that each method that enforces certain security feature is associated with a method the removes or verifies the added security. For example, a method encrypting data must be paired with a method for decrypting the data. A method for calculating a MAC must be paired with a method for verifying the MAC.

Transformational security is encapsulated into two method calls per subobject. First method adds the security that the other method removes or verifies. Security-adding methods are called by the sending local object and the security-removing and verifying methods by the receiving local object. The general calling sequence (Fig. 4), is as follows:

1. A client accesses the local object through the control subobject.
2. The control subobject marshalls the arguments to the method call and prior to passing the marshalled arguments to the replication subobject, calls the SecCtrlAdd() method of the security subobject. The method does all the security related processing relevant to the control subobject and adds the required fields into the argument string.

3. The control subobject passes the processed argument string to the replication subobject.
4. The replication subobject processes the marshalled arguments and invokes the SecReplAdd() method that implements the security relevant processing and adds the necessary data to the arguments.
5. The data is passed to the communication subobject.
6. The communication subobject processes the data and calls the SecCommAdd() method that implements the required features of communication security prior to sending the resulting data through the communication channel.
7. The data is communicated to the other local object.
8. Upon receipt of data, the receiving communication subobject calls the SecCommRemove() method of the security subobject that reverses and verifies the security measures put on place by the SecCommAdd() method.
9. The data is processed by the communication subobject and passed to the replication subobject.
10. The replication subobject calls the SecReplRemove() method that reverses and verifies the security measures put on place by the SecReplAdd() method.
11. The replication subobject processes the data and passes it to the control subobject.
12. The control subobject calls the security subobject method SecCtrlRemove() that reverses and verifies the security measures put on place by the SecCtrlAdd() method.
13. The control subobject proceeds with the method invocation. The results are passed to the remote client and the security process is repeated.

Some security methods may be NULL as there may not be security processing at each subobject. However, it is imperative that the methods are called in the above sequence to enable replacement of the security subobject without modifying other subobjects.

The interface to the security subobject is simple, and the semantics of the byte strings passed as arguments to the methods are determined by the conventional subobjects. Security processing does not need to be aware of it.

Each method must receive the arguments through reference and throw a GlobeSecurityException or return a GlobeSecurityError code, depending on the implementation language. This enables subobjects to standardize the handling of security errors.

5.2 Access Control

Access control is concerned with which methods of a local object can be invoked by which clients under which conditions. For example, access to Write() methods may be restricted to the members of the core group, and may only be granted after strong authentication and if a certain environmental condition, such as time of the day, is met. This requires a sophisticated access control scheme to mediate method invocation. This is logically placed in the control subobject.

Access control functionality can be divided into two layers: credential verification layer and access enforcement layer. As the access control decisions are made at the control subobject, there is a need to store the results of credential verification in a data structure available for the access control decision function.

The security subobject maintains a Credentials data structure that consists of a number of *(attribute, value)* pairs, where the values of different attributes are set by the security subobject once credentials are verified. The access control decision function reads the credential data structure, access control rule base, and a set of environmental variables needed in access decisions. The access control facility can be invoked by the CtrlVerifyAccess(methodID) method, prior to step (13) in Fig 4.

The access control rule base consists of a number of authorization statements describing under which conditions certain clients are allowed to invoke certain methods of the semantics subobject.

5.3 Secure Operations

Similar methods than those required for security in method invocation are required for control messages. Similar to method invocations, control messages need to be protected in transmission, and a right to invoke certain control operations may be restricted to certain local objects. Separate security associations may need to be maintained for control messages.

Previous discussion has focused on the secure invocation of methods remotely. There are also security requirements that are not related to method invocations. These requirements are mostly concerned with the dynamic aspects of DSOs, most importantly the binding of new local objects to a distributed shared objects.

Each Globe object has a unique ID. The naming service maps a symbolic name to an object ID. The location server can then be queried for the contact address of a local object to bind to. Prior to the actual binding, an object class must be retrieved from the local implementation repository, and the local object constructed from the object class.

Name servers, location servers, and implementation repositories may not belong to a single administrative domain. Therefore, clients may not equally trust all service providers. Measures are required for adequate security at all the services. Different from, say, security extensions to the Internet Domain Name Server (DNS), naming and location information may not always be public. Therefore, it is unlikely that existing standards can be directly applied in Globe.

Research is currently carried out to investigate the extent to which existing infrastructure services can be applied in Globe.

6 Conclusions

This paper has analyzed the difficulties in designing security of DSO platforms using Globe as a reference system. In particular, the objective of isolating security relevant processing from other computations constitutes a fundamental

design challenge. Yet, it is essential to enable a framework where appropriate assurance of the correctness of security design and implementation can be achieved.

We have concluded that, despite certain disadvantages, it is better to centralize security enforcement into a single security subobject. The method names and calling sequences from subobjects can then be standardized to enable replacement of the security subobject without modifying other subobjects.

The interface of the security subobject consists of three types of methods. Transformational security measures are used for protecting method invocations and control messages during communication. Access control methods prevent unauthorized clients from invoking methods or unauthorized local objects from invoking control methods. Other security measures are applied for other security operations, such as secure binding.

The work is currently on progress, and certain applications have been developed in Globe and different more advanced applications scenarios are currently under research. Further research is also going on in the provision of a high level policy-mechanism independence.

As the Globe is currently implemented in Java, there are certain possibilities for replacing the implementation on-line, for example during the binding of a new client to an existing local object.

References

1. M. D. Abrams, H. J. Podell, and D. W. Gambel. Security engineering. In *Information Security, An Integrated Collection of Essays*, volume Abrams, Marshall D. and Jajodia, Sushil and Podell, Harold J., pages 330–349. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.
2. A. Anderson, D. Longley, and L. F. Kwok. Security modeling for organizations. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 241–250. ACM Press, 1994.
3. A. Bakker, M. van Steen, and A. S. Tanenbaum. From remote objects to physically distributed objects. In *Proceedings of the 7th IEEE Workshop on Future Trends of Distributed Systems*, 1999.
4. K. P. Birman. *Building Secure and Reliable Applications.* Manning Publications Corporation, Greenvich, CT, USA, 1996.
5. D. L. Brinkley and R. R. Schell. Concepts and terminology for computer security. In *Information Security, An Integrated Collection of Essays*, volume Abrams, Marshall D. and Jajodia, Sushil and Podell, Harold J., pages 40–97. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.
6. S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security.* Addison–Wesley, Wokingham, UK, 1995.
7. International standard ISO/IEC 15408 common criteria for information technology security evaluation (parts 1-3), version 2.0, CCIB-98-026, May 1998.
8. D. Gollmann. *Computer Security.* John Wiley & Sons, Chichester, UK, 1999.
9. W. Kou. *Networking Security and Standards.* Kluwer Academic Publishers, 1997.
10. R. Kruger and J. Eloff. A common criteria framework for the evaluation of information technology security. In *Proceedings of the IFIP TC11 13th International Conference on Information Security, (SEC'97)*, pages 197–209. Chapmann & Hall, 1997.

11. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, NY, USA, 1997.
12. R. Oppliger. *Authentication Systems for Secure Networks*. Artech House, Norwood, MA, USA, 1996.
13. C. Salter, O. Saydjari, B. Schneier, and J. Wallner. Toward a secure system engineering methodology. In *Proceedings of the New Security Paradigms Workshop*. ACM Press, 1998.
14. R. C. Summers. *Secure Computing: Threats and Safeguards*. McGraw–Hill, 1997.
15. M. Van Steen, P. Homburg, and A. S. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency*, pages 70–78, January–March 1999.
16. W. A. Wulf, C. Wang, and D. Kienzle. A new model of security for distributed systems. In *Proceedings of the ACM New Security Paradigms Workshop*, pages 34–43, Lake Arrowhead, CA, USA, 1996. ACM Press.

This article was processed using the LaTeX macro package with LLNCS style