Chapter 1

Introduction

The object-oriented paradigm continues to emerge as a useful and unifying one in computer science and engineering. Ideas from the paradigm have been incorporated in such diverse fields as software engineering, artificial intelligence, and database management, with the resulting advancement of these fields in new directions. The interest of the artificial intelligence community to object-orientation has come from the search for knowledge representation and management schemes for large knowledge bases. From the software engineering perspective, objects are seen as instances of abstract data types (ADT's) [LZ74] incorporating the principles of encapsulation and information hiding. It is now well known that these are vital in building software systems and components that are highly modular, maintainable, and reusable. From the database viewpoint, the object-oriented data model overcomes the limitations of record-based models by being able to model complex structures, relationships, and behavior. These capabilities are essential for emerging application areas such as computer-aided design and manufacturing (CAD/CAM), software management and reuse, and office information systems, to name a few.

Central to the object-oriented paradigm is the notion of an object. An object models some real-world entity, encapsulates some private state, and is identified by some persistent and unique identity. An object also encapsulates services which are made available to requesting clients through a public interface. Services are
implemented by pieces of code called methods, and invoked by sending messages to objects.

Although there is wide agreement on the above primitive notion of objects, much debate has ensued in the last few years over the formulation of an object-oriented data model. There is some consensus on the core concepts that such a data model should support. However, a single universal definition has not emerged [Mai89]. Some researchers also distinguish between object-based, class-based, and object-oriented systems. In Wegner's view [Weg87], a language is considered to be object-based if it provides linguistic support for objects. Examples of such languages include Actors [Agh87], and Ada. In addition, if a language supports the notion of classes, it is considered to be class-based. An example of such a language is Clu [L+77]. Further, if the language supports the notion of inheritance, it is categorized as being object-oriented. Inheritance allows objects and classes to share structure in terms of inherited attributes and behavior in terms of inherited methods.

In this dissertation, we address the support of object-based computing in multilevel-secure environments. Such environments are characterized by objects labeled at different classifications, and accessed by users (or more precisely subjects on their behalf) cleared to various security levels. A security policy governs how the subjects can access the various objects in the system. We are not concerned with the plethora of modeling issues and variations in the object-oriented data model. Rather, we are interested in how objects can form a basic model for secure computing. Computing in the object framework reduces to sequences of message passing and method invocations among objects. We are thus interested in how objects can send messages and exchange information without violating the relevant security policy.

It is now widely recognized that computer and information security consists of three distinct but interrelated areas, namely confidentiality, integrity and availability.
Confidentiality is concerned with the disclosure of information, integrity is concerned with the improper modification of information, and availability is concerned with the denial of access to information. In this dissertation our focus is primarily on confidentiality and integrity issues.

Before proceeding further in the discussion, it is helpful to clarify the distinction between users, principals, and subjects. By a user, we mean a human being, represented in the system by a unique user identity. Each user may be associated with several principals. However, a principal can be associated with only a single user. Each principal associated with a single user may be given a different set of access rights, and allowed to login at different security levels as long as these levels are dominated by the clearance of the parent user. Each principal may in turn have several subjects. Each subject is a process in the system.

The rest of this introductory chapter is organized as follows. The first section discusses concurrency, message passing, and other issues in object-based computing that are relevant to the topic of this dissertation. This is followed by a brief introduction to multilevel security. The last section highlights the organization of the rest of the dissertation.

1.1 Object-Based Computing

The wide applicability of the object-oriented paradigm and way of thinking has naturally led to the integration of many technologies. For example, a noticeable trend is the enhancement of object-oriented programming languages with support for objects that are persistent and sharable. The ability to support persistent and sharable objects is indeed the main feature offered by database management systems. Thus the distinction between object-oriented programming environments and databases is increasingly becoming blurred. Another wave of integration will likely result from
concurrent object-based computing environments incorporating database functionality. Application areas such as situation assessment, network monitoring, and process control are naturally modeled as cooperating concurrent objects. However, the states of such objects may be required to persist beyond individual computing sessions and process lifetimes, and to be shared by other objects. For example, in a process control system, persistency may be required for the simple reason that we need history information for measurements of quantities such as temperature and pressure.

The issues addressed in this dissertation are most pertinent to domains which can be modeled as a set of autonomous but cooperating objects such that the object-based model of computing is a good fit, and where object states are required to be persistent and sharable. The mechanisms we propose can be incorporated into a variety of multilevel computing environments and systems such as object-oriented databases and message-based operating systems.

1.1.1 Concurrency and Message-Passing Alternatives

The object-based model of computing sees the world as a set of concurrent and cooperating objects. As such these objects would have to occasionally interact with each other and exchange information. An obvious approach would be to use common variables residing in some shared memory. However object interaction based on message passing is the most suitable, when objects are autonomous entities executing in a loosely coupled environment. We may categorize message passing alternatives into two categories:

- **Synchronous.** In the synchronous form of message passing, a sender object $O_1$ sends a message to a receiver object $O_2$, and is suspended until the message is delivered to the intended receiver and a reply returned.
• Asynchronous. In this case the sender transmits a message and continues execution without waiting for the message to be delivered, or for the reply.

Synchronous message passing essentially parallels the semantics of remote procedure calls (RPC's), with the difference that the receiver's activity does not have to end with the return of the reply. The asynchronous form varies in style from the sending of a single message to more complex forms involving streams of messages. In either case, asynchronous message passing requires synchronization if the sender needs to access the reply returned from the receiver. This is because the sender and the receiver may be executing concurrently. The well-known approach to such synchronization is based on the notion of futures [Weg91]. A future is a data structure (object) that represents the results of the concurrently executing receiver object (process). When a message is sent in asynchronous mode, a future is created, after which the sender continues execution. The future represents a promise or I.O.U. from the receiver object (the called process). When the sender subsequently wishes to obtain the reply it accesses the future object. If the promise has not been fulfilled, the sender has to wait until the receiver returns the required results to the future object.

In the multilevel context, whenever a sender has to wait for a reply or some other result, there always exists the possibility of confidentiality leaks through covert channels (as discussed in subsection 1.2.2). This may occur from both synchronous and asynchronous communications. However, in our subsequent discussions, we focus on synchronous (RPC-based) communications as the intended correctness semantics is clear.

1.1.2 Persistent and Sharable Objects

When objects persist beyond individual user sessions and are shared by many users, the integrity of the data objects becomes a primary concern. The actions or updates
of one user on an object can causally influence the values read and written by other users. Further, information can flow from one object to another.

Several definitions of data integrity have appeared in recent literature. Sandhu has succinctly compared and summarized these different notions of integrity [San93]. Of these different notions of integrity, the one most suitable in our context is that which is concerned with the improper modification of data. Our objective is to make sure that objects are modified in a fashion that is consistent with a synchronous message passing (remote procedure-call) semantics. We are not concerned with data integrity in the sense of an expectation of data quality that may incorporate liveness requirements.

1.2 Multilevel Security

The notion of multilevel security for data confidentiality originated in the late 1960's when the U.S. Department of Defense wanted to protect classified information processed by computers. Environments and applications requiring multilevel security are characterized by users with more than one clearance level sharing data with more than one sensitivity level (classification).

In the following subsections, we review basic notions of multilevel security by introducing lattice-based security policies and models. We then discuss how even with adequate access control mechanisms, information may still leak through covert and signaling channels.

1.2.1 Lattice-based Security Models

The military security policy is a special case of a more general lattice-based security policy. Every object in the system is assigned a security class (also known as a security
label). Information is allowed to flow between two objects only if the policy allows information to flow between the corresponding classes. Given a set $SC$ of security classes, we can formally define a binary can-flow relation $\rightarrow \subseteq SC \times SC$. It is also convenient to define the inverse of the can-flow relation called the dominates relation. We say $A \geq B$ ($A$ dominates $B$) if and only if $B \rightarrow A$.

In a lattice-based approach to multilevel security, the security classes form a mathematical structure called a lattice. The security elements of the lattice are partially ordered under the can-flow ($\rightarrow$) relation. In a lattice, there may be pairs of elements say $(A, B)$ for which the can-flow relation does not hold (i.e., $A \not\rightarrow B$ is true). Every pair of elements in a lattice possesses a greatest lower bound. In other words, for every pair of elements $(A, B)$ there is an element $L$ such that $L \rightarrow A$ and $L \rightarrow B$ hold. Similarly, every pair of elements in a lattice has a least upper bound.

In the military and government setting, the security label given to a data item (object) consists of two parts, a hierarchical level and a category. The set of hierarchical levels is totally ordered as follows:

Top-secret $>$ Secret $>$ Confidential $>$ Unclassified

The individual set elements in a category are known as compartments. Compartments are used to implement the principle of least privilege or "need-to-know". This is a well known principle in security and ensures that a user has access to the minimum number of objects required to perform his or her job. The compartment assigned to an object typically reflects the subject matter of the information contained in the object. We say a security level dominates ($\geq$) another if its hierarchical level is greater than the other's and its category set includes the others. As an example consider two compartments CRYPTO and ATOMIC. Thus the level <top-secret; CRYPTO, ATOMIC> dominates the level <secret; CRYPTO>.

Having introduced lattice-based security policies, we now turn our attention
to related security models. A security model is used to implement a security policy. The Bell and LaPadula security model (also called the BLP model) was the first to formally address multilevel security, and even today remains the *de facto* standard [BL76]. BLP characterizes and governs access control and information flow with the following two rules (*l* denotes the label of the corresponding subject (*s*) or object (*o*)).

- **Simple Security Property.** Subject *s* can read object *o* only if *l*(s) ≥ *l*(o).

- **←-Property.** Subject *s* can write object *o* only if *l*(s) ≤ *l*(o).

The need for the simple-security rule is obvious; it prevents low level users (and subjects) from reading information stored at higher levels. It thus prevents "read-up" operations. This requirement parallels that of the paper world with documents and human beings (users). However, it turns out that disallowing read-up operations alone is not sufficient to prevent illegal information flows that violate the security policy. To illustrate, a high subject may read a file classified at high, and write a subset of its contents (or information derived from its contents) into a second file at a lower file. This would clearly violate the security policy as information is flowing downwards in the security lattice. The ←-property (pronounced star property) prevents such violations by disallowing write-down operations.

The ←-property is really a confinement property and at first sight might appear to be overly restrictive and counter-intuitive to the way the real (paper) world works. A human user cleared to a high level may talk and disclose information to users at lower levels. However, in the real world, the high user is "trusted" to exercise judgement and not disclose any highly classified information. In other words the user is allowed to talk about public information such as sports scores and the weather. A subject, which is a program or process in the system, cannot be trusted in a similar fashion.
The above distinction between the trust placed in human users and the mistrust of subjects is significant and fundamental to implementing security policies in computer systems. A subject (computer program) cannot be trusted as it may contain bugs and trojan horses. A trojan horse is a malicious program which in addition to its stated objectives, performs some hidden functions. Trojan horses are typically embedded into application programs and utilities. A typical victim such as an end user, is not aware of the Trojan horse when using the infected program. Fig 1.1 illustrates how information could be leaked with the help of a Trojan horse. A high subject (program) reads two high level files and compiles a high level report. Meanwhile, the Trojan horse that is embedded in the program, writes some information in the high level files to a low level file. The $\approx$-property prevents such leaks.

The Bell-LaPadula rules are examples of what we call mandatory access control (MAC) rules. Mandatory access control places restrictions on the access of objects based on their security labels, and the controls themselves cannot be bypassed. In many models and systems (including BLP), mandatory access control is complemented by discretionary access control (DAC) mechanisms. In the DAC framework, objects are owned by subjects who at their discretion, propagate rights to these objects, to other subjects. An example of a discretionary access control rule would be one that specifies how the owner of a directory may give read or write permission to the directory, to other subjects. In this dissertation we do not address discretionary
access control issues as they are irrelevant to the problems associated with supporting write-up actions.

1.2.2 Covert and Signaling Channels

In our discussion so far, we have seen how mandatory access controls prevent illegal communication (information flows) between access classes. However, a system that enforces mandatory access controls such as one based on the BLP model can be riddled with covert channels (also called leakage paths). These channels arise due to processes sharing resources in the system, and pose a formidable problem in building secure systems. Models such as BLP approach access control from a certain level of abstraction. However, there exists several shared resources and variables such as buffer pools and global counters that are not part of the abstractions of BLP, but nevertheless are shared by processes (subjects) at multiple security levels.

There exists two types of covert channels. A storage channel arises when a process writes an object or variable and another process can observe or read the effect of the write. A timing channel results when the activity of a high level subject affects the performance of the system in such a way that it can be observed and measured by lower level subjects. To exploit timing channels, subjects must have the ability to measure time (such as having access to a real-time clock). Storage channels do not require access to any such timing base.

As an illustration, let us see how a storage covert channel based on resource exhaustion can be formed. Consider a computer system that has a 10MB pool of dynamically allocated memory. A high subject requests the entire pool of memory and its request is granted. This is followed by a request from a low subject for 10MB of memory. Obviously, this request cannot be granted, and the low subject records this fact as one bit of information. Now a colluding high subject can selectively
request and release the pool of memory at regular intervals, causing the low subject's request to be denied or honored in a specific pattern, and thereby opening up the covert channel. Another example of a storage channel is one that arises from a low subject inferring the existence of an object at a higher level. A low subject may accomplish this by attempting to create a file with a file-name that already exists, and the file system rejecting the request since it has to guarantee the uniqueness of file names. A pair of colluding subjects can cause a pattern of bits to be leaked. Lastly, a simple illustration of a timing channel is one that is caused by the modulation of CPU utilization. A high subject can vary the CPU utilization at some constant interval, causing the low subject's progress to be modulated and measured.

In summary, a covert channel is a communication channel not normally intended for direct communication between subjects in the system. These channels are beyond the purview of abstract security and access control models such as BLP. Detecting and closing them will require analysis of information flow within the internals of individual systems.

In this dissertation we distinguish between the above mentioned covert channels and signaling channels [JS91]. A signaling channel is a means of downward information flow which is inherent in a data or computation model, and will therefore occur in every implementation of the model. A covert channel on the other hand is a property of a specific implementation, and not a property of the data or computation model. In other words, even if the data or computation model is free of downward signaling channels, an implementation may well contain covert channels due to implementation specific quirks. Conversely, closing all covert channels in an implementation will not eliminate signaling channels. It should be noted however that both covert and signaling channels form illegal and unintended communication paths.
As mentioned before, the object-based computation model essentially reduces to sequences of message passing among objects. Unfortunately, with such a computation model, message passing with remote procedure-call (RPC) semantics is vulnerable to signaling channel attacks, as we will see shortly. This has broad implications for building multilevel-secure systems that support object-based computing. In particular, solutions to close signaling channels should be addressed to the layer that supports the data and computation model, rather than fine-tuning low-level implementation parameters. The impasse formed by signaling channels appears to be fundamental to object-based computing, and is indeed one of the main issues that we address in this dissertation.

1.2.3 Security Kernels, Architectures, and Trusted Subjects

Are there approaches to building secure systems that overcome the security problems inherent in conventional design methodologies? One answer to this lies in the idea of building secure operating systems with a security kernel approach. The lower-level functions in an operating system are generally performed by what is considered to be the kernel (also called the nucleus or core). Similarly, in a secure system the security kernel implements the security mechanisms of the operating system. The security kernel approach is based on (and implements) the concept of a reference monitor. The reference monitor, which is the combination of hardware and software, enforces the security policy by providing access control to resources. It utilizes access control information stores in an access control database for this purpose.

The successful application of the security kernel approach is based on the theory that only a small fraction of the total functions in an operating system are needed to enforce security. The motivations to isolate the security functions into a security kernel are many. Isolation makes it easier to protect, modify, as well as verify
these security mechanisms. It also makes it easier to ensure completeness or coverage; i.e., every access to a protected object must pass through the security kernel.

Several systems that employ the security kernel approach also support the notion of trusted subjects. The most distinguishing feature of trusted subjects is that they are endowed with certain privileges. Most notably, a trusted process may be allowed to bypass mandatory security controls. Thus a trusted subject could access information at various security levels and be allowed to write-down such information (as it is exempt from the *-property restrictions of mandatory security).

There has been a certain degree of controversy, uneasiness, and suspicion with the notion of trusted subjects. One of the major drawbacks of utilizing trusted subjects has to do with the greater difficulty in verifying trusted software. The properties and associated proofs of trusted subjects are not as obvious or straightforward as that for the simple security and * properties. In the remainder of this dissertation, we refer to architectures that use trusted subjects as trusted-subject architectures. Architectures that use only single-level (untrusted) subjects are referred to as kernelized architectures.

1.2.4 Multilevel System Architectures

In this section we briefly review three multilevel database management system (DBMS) architectures, namely, the kernelized, replicated, and trusted subject architectures. Of these, the first two comprise two of the three architectures identified by the Woods Hole study organized by the U.S. Air Force [Cou83]. These architec-

---

1 The term "trusted" is used often in the literature to convey one of two different notions of trust. In the first case, it conveys the fact that something is trusted to be correct. In the second case, we mean that some subject is exempted from mandatory confidentiality controls; in particular the simple-security and *-properties in the Bell-Lapadula framework. It is the latter sense of trust that we refer to here.
Figure 1.2: A kernelized multilevel system architecture

Figure 1.3: A multilevel system architecture with trusted subjects
Figure 1.4: The replicated multilevel system architecture

tures were motivated by the need to build multilevel secure DBMS's from existing untrusted DBMS's. The trusted subject architecture on the other hand, requires one to build a multilevel DBMS from scratch.

The basic features of the kernelized architecture are shown in figure 1.2. Most noticeable is the fact that there exists an individual DBMS for every security level. A user cleared to level high, interacts with the high DBMS through a trusted front end. Existing DBMS's can be directly incorporated with minimal modification since they have to manage only single-level data at their levels. These single-level DBMS's communicate with a trusted operating system that manages both high and low data storage.

The trusted subject architecture differs from the kernelized one in that we no longer have single-level DBMS's for every level, as shown in figure 1.3. Instead, there exists a single (trusted) DBMS that incorporates multilevel trusted subjects. Recall that such a subject is exempt from the mandatory access control rules enforced by
the operating system.

In contrast to the kernelized and trusted subject architectures, the distinguishing feature of the replicated architecture shown in figure 1.4 is the existence of separate DBMS’s for every level and the replication of low data at the higher level DBMS’s. The high assurance of this architecture stems from the fact that the DBMS’s are physically isolated and a subject cleared to a level, say $l$, is allowed to access only the database for level $l$, and can obtain all the data needed at the local DBMS location.

1.3 Multilevel Security in Object-based Computing

In this section, we discuss the integration of multilevel security in object-based systems. In attempting such an integration, are there fundamentally different approaches to providing access control and authorization in object-based systems? Keefe [Kee90] has provided a useful framework and categorization of existing object-based security models and access control approaches. Namely, these include behavior-based, structure-based, and message-based approaches.

An object models the behavior of an entity through the methods supported by its interface. Accordingly, a behavior-based approach to multilevel security specifies and enforces access control in terms of method invocations. Thus the access control problem essentially reduces to the question: Is subject $S$ allowed to invoke method $M$ on object $O$? Access control and rights is no longer seen in terms of read and write operations; rather at the higher and semantic level of abstract operations. The idea of behavior-based access control meshes well with the notion that objects are instances of abstract data types. Unfortunately, the semantic and abstract nature of this approach gives no clue on how to construct a more concrete and implementable model.

In the structure-based approach, access control is no longer seen in terms of
the semantics of methods or abstract operations. Rather, it is based on mediating primitive read and write operations issued by methods. The mediation of reads and writes essentially reduces this approach to an interpretation of the Bell-LaPadula (BLP) model for objects. Access to portions of object states by subjects, is governed by the simple-security and +-properties of BLP.

The behavior-based and structure-based approaches reflect the bias which systems have in supporting behavioral and structural features. Dittrich has provided a useful taxonomy for object-oriented databases [DHP89], and observes that behaviorally object-oriented systems tailor all mechanisms to emphasize the support of abstract data types. Structurally-oriented systems emphasize the access and manipulation of complex and nested object structures. Fully object-oriented systems provide both behavioral and structural features.

A third approach, referred to hereafter as the message or flow-based approach, is based on the central notion that in the object-based model, objects communicate with each other solely through messages. Since objects are encapsulated units, it follows that security can be enforced by controlling the exchange of messages. When a message is sent, the classifications of the sender and receiver objects are used to determine if an illegal information flow will take place. The message is delivered to the receiver only if the resulting information flow does not violate the security policy. It is important to note that there is no attempt to analyze the semantics (i.e., message type) or contents of the message itself.

How does the message-based approach compare with the behavior-based and structure-based approaches? In some sense, by ignoring the semantics of messages the message-based approach appears to be less rich than the behavior-based one. Nevertheless, it offers the advantage of meshing well with the object-based model of computing. Thus it has wide applicability in providing security for systems ranging
from message and object-based operating systems to object-oriented databases. A further advantage of the message-based approach is that it lends itself easily to an implementation. It is conceptually simple and elegant to enforce security by mediating messages at a central point such as the trusted computing base. When compared to structural approaches, the message-based approach incurs less overhead since access control need not be enforced on the many primitive read and write operations that can be issued by methods.

Another issue that arises with the integration of multilevel security and object
notions, is that of object classification granularity. What is the basic size of the unit that should be classified independently from other units? If the granularity is larger than an object, the system may offer good performance. However, from a modeling standpoint this may not be flexible enough to classify the objects accurately and naturally. If the granularity is very fine and smaller than an object, the system may offer great modeling flexibility, although at the price of performance.

Figure 1.5 illustrates five different approaches and associated granularities for object classification discussed in [Kee90]². In the first approach, referred to as instance variable value labeling, every instance variable or element of an object is assigned an independent security level. In the second approach, called object labeling, the entire object is uniformly classified. A variation of this second approach yields a third one called partial object labeling, in which all the instance variables share a common uniform classification which may be different from the rest of the object. The fourth approach called class labeling, results in every object instantiated from a class, to be object labeled with a single common classification. The last approach referred to as instance variable labeling, requires the classification of the object-identifiers for all the instance objects instantiated from a class, to be the same. Further, all the instance variables are required to have the same label, or labels that fall within some prespecified range.

In summarizing this section, we note that the approaches to authorization, and classification granularity, are related as they influence each other. The primitive read and write operations always apply to the values of individual instance variables of an object. Thus instance variable value labeling lends itself to a structure-based access control. On the other hand, if message or flow-based access control is used, it is more natural to classify objects uniformly. On receipt of a message, the corresponding

²It is of course possible to enumerate other combinations and possibilities.
method invoked is generally given access to the entire object state within the object boundary. Thus object labeling appears to be more suitable.

1.4 Summary of Previous Work

In this section we review some of the existing proposals to integrate multilevel security in object-based and object-oriented systems. The motivation for most of these efforts have come primarily from object-oriented databases.

A behavioral approach to access control is pursued in [MH89]. A subject $S$ is allowed to invoke a method $M$ on an object $O$, if a relationship exists between the subject, object, and the method. The security policy is expressed as a set of such relationships.

Structure-based approaches are mentioned in [KTT88, ML92, Lun90, Thu89a, Thu89b]. All these approaches accommodate fine-grained classification granularity by supporting multilevel objects (objects where each component is classified independently). Thus access to components of objects is governed by mandatory access control rules. These models also consider illegal information flows that could occur from classes to instances, as well as through inheritance along the class hierarchy. In order to prevent such flows, these models require a number of constraints to be maintained. For example in [Thu89b] it is required that the classification of an instance of a type dominate the classification of the type. In [Lun90], the hierarchy property requires that the classification of a subclass dominate the classification of the parent superclass. This ensures that information flow along the class hierarchy is always upwards in the security lattice.

Flow or message-based approaches to multilevel security were initially proposed in [BTMD89, CVW+88] for message-based secure operating systems. Similar ideas for databases and information systems are mentioned in [MO87, TC89, JK90].
The underlying theme in all these proposals is to enforce security by mediating message flow between objects. The model in [JK90], referred to hereafter as the message filter model, calls for a message filter component to filter messages. The message filter here is the analog of the reference monitor.

Our discussion above has been intentionally brief as the focus in this dissertation is on a very specific issue: the support for write-up actions by sending messages upwards in the security lattice. While the modeling flexibility and ease of implementation of the above proposals and models vary, the support for write-up actions is generally absent, or if present, not worked out in sufficient detail.

The models in [ML92, Thu89b] explicitly prohibit write-up actions. The original message filter proposal in [JK90] allows messages to be sent upwards in the security lattice and places no restrictions on write-up actions. However, it does not address the details and complications involved in doing this. The model in [ML92] also allows messages to be sent upwards in security levels. As in the message-filter model, the authors note that the actual replies to such messages cannot be returned to the lower level senders, and hence have to be substituted with innocuous NULL or NIL messages. However, the following observation by the authors is significant:

"Note that the system, rather than the higher-level subject, should determine the time it takes to deliver the null value, otherwise a timing channel will exist. The underlying TCB is responsible for this protection."

But how will the TCB provide such protection? What are the architectural requirements? In providing such protection, will integrity be compromised? It is precisely answers to these specific questions that are pursued in this dissertation.

1.5 Organization of Thesis
<table>
<thead>
<tr>
<th></th>
<th>Trusted Subject</th>
<th>Kernelized</th>
<th>Replicated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity</td>
<td>Serial correctness</td>
<td>Serial correctness</td>
<td>Serial correctness</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Final-state-equivalence</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Session manager is noninterfering</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1: Summary of main results

The major results of this dissertation cast in terms of the various theorems are summarized in figure 1.1. In the kernelized and replicated architectures we need to show that our solutions preserve integrity. In the trusted subject architecture, we need to show that in addition to integrity, our solutions do not introduce any confidentiality leaks.

The rest of this dissertation is organized as follows. Chapter 2 motivates the main problem addressed in the dissertation. Chapter 3 is a quick overview of the message filter security model, while chapter 4 presents our asynchronous computation model. Chapters 5, 6, and 7 discuss the implementation of our ideas in trusted subject, kernelized, and replicated architectures, respectively. Chapter 8 discusses inter-session synchronization schemes, and chapter 9 summarizes the dissertation and also highlights some future directions for research.