

Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-11-2008

Composable Distributed Access Control and Integrity Policies for Query-Based Wireless Sensor Networks

David W. Marsh

Follow this and additional works at: <https://scholar.afit.edu/etd>



Part of the [Digital Communications and Networking Commons](#), and the [Information Security Commons](#)

Recommended Citation

Marsh, David W., "Composable Distributed Access Control and Integrity Policies for Query-Based Wireless Sensor Networks" (2008). *Theses and Dissertations*. 2635.
<https://scholar.afit.edu/etd/2635>

This Dissertation is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.



COMPOSABLE DISTRIBUTED
ACCESS CONTROL AND INTEGRITY POLICIES
FOR QUERY-BASED WIRELESS SENSOR NETWORKS

DISSERTATION

David Wesley Marsh, Major, USAF

AFIT/DEE/ENG/08-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/DEE/ENG/08-06

COMPOSABLE DISTRIBUTED ACCESS CONTROL AND INTEGRITY POLICIES FOR
QUERY-BASED WIRELESS SENSOR NETWORKS

DISSERTATION

Presented to the Faculty
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

David Wesley Marsh, BSEE, MSCE
Major, USAF


March 2008

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

COMPOSABLE DISTRIBUTED ACCESS CONTROL AND INTEGRITY POLICIES FOR
QUERY-BASED WIRELESS SENSOR NETWORKS


David Wesley Marsh, BSEE, MSCE
Major, USAF

Approved:



Dr. Rusty O. Baldwin (Chairman)

5 Mar 08
Date



Dr. Yung Kee Yeo (Dean's Representative)

6 Mar 08
Date



Dr. Barry E. Mullins (Member)

5 Mar 08
Date



Dr. Robert F. Mills (Member)


5 MAR 08
Date



Dr. Michael R. Grimaila (Member)

5 MAR 08
Date

Accepted:



M. U. Thomas
Dean, Graduate School of
Engineering and Management

11 Mar 08
Date

Abstract

In much the same way as the Global Positioning System (GPS) was initially intended as a navigation system for military use, current Wireless Sensor Networks (WSNs) tend to be dedicated to a particular purpose and for the exclusive use of a particular organization. Thus, WSN security has focused in large part on encryption and authentication schemes to protect data during transmission and ensure only authorized users are granted network access. However, just as GPS has become an indispensable resource for military and civilian use, it is expected that WSNs will become an essential sensing resource that must support a vast number of users.

This will require security measures mirroring the security concepts of typical secured computer networks in which authenticated users are only authorized to access certain data within the system. WSNs will need to similarly restrict access to data, enforcing security policies to protect data within WSNs, even though WSN nodes face severe power, memory, computational, and communication limitations. To date, WSN security has largely been based on encryption and authentication schemes.

The WSN Authorization Specification Language (WASL), a mechanism-independent composable WSN policy language, takes into account the severe resource constraints of WSN nodes. The language is itself specified and implemented with a JavaCCTM grammar-parser for security policy input and a policy compiler built using JavaTM code.

WASL is capable of specifying arbitrary and composable security policies that span and integrate multiple WSN policies. The construction, hybridization, and composition of well-known models is demonstrated to preserve security, sustaining confidentiality in Bell-LaPadula's model, integrity in Biba's strict integrity model, and conflict of interest avoidance in the Chinese Wall.

Using WASL, a multi-level security policy for a 1000 node network requires only 66 bytes of memory per node using a naïve data compression scheme. A policy of this size can reasonably be distributed throughout a WSN periodically. The compilation of a variety of policies and policy compositions are shown to be feasible using a notebook-class computer not unlike that expected to be performing typical WSN management responsibilities.

A system implementing WASL is secure as defined by the security model. It is also be more flexible in that a policy file update is all that is required to modify the accesses permitted any given user. The policy can be additionally modified to permit inter-network accesses with no more impact on the WSN nodes than any other policy update.

To My Family

Acknowledgements

The success of this dissertation does not belong solely to me. Dr. Rusty Baldwin was an incredible mentor as my research advisor and chairman of my committee, helping me work through the process of conceiving, developing, and demonstrating my work. His insightful reviews of my written documents were particularly helpful in identifying areas requiring further examination. Each member of my committee, Dr. Barry Mullins, Dr. Robert Mills, and Dr. Michael Grimaila, used his expertise to help me evaluate the quality and impact of the decisions I made and the documents I wrote. Dr. Robert Graham and Dr. Thomas Hartrum educated me in the areas of software engineering and formal methods and Timothy Halloran taught me software analysis techniques; without these skills I could not have accomplished this work.

Each one mentioned above played an important role in my academic growth and has my sincere gratitude.

David W. Marsh

Table of Contents

	Page
Abstract	iv
Acknowledgements	vi
List of Figures	ix
List of Tables	x
I. Introduction	1
1.1 Scenarios Identifying a Technology Gap	1
1.2 Background	3
1.3 Purpose	4
1.4 Summary	7
II. Technical Area Review	8
2.1 Wireless Sensor Networks	8
2.1.1 Limitations	8
2.1.2 Communication	9
2.1.3 Query-Based Network	10
2.1.4 Security Issues	11
2.1.5 TinyOS	13
2.2 Computer Security	14
2.2.1 Fundamentals	15
2.2.2 Confidentiality	16
2.2.3 Integrity	20
2.2.4 Chinese Wall Model	22
2.3 Security Policy Representation and Composition	25
2.3.1 Introduction to Security Policy Composition	25
2.3.2 A Policy Language	26
2.3.3 A Policy Composition Framework	32
2.3.4 An Policy Composition Algebra	33
2.4 Related Research	37
III. WSN Authorization Specification Language	39
3.1 Network Under Study	40
3.1.1 System Characteristics	41
3.1.2 Distribution of Responsibilities	41
3.2 WASL: The Language	43
3.2.1 Grammar	43
3.2.2 Declarations of Terms	43
3.2.3 Relations	45
3.2.4 Rules	46
3.2.5 Compilation	47
3.3 Summary	49

	Page
IV. Specifying Policies in WASL	50
4.1 BLP, Biba, and CW Rules in WASL	50
4.2 Creating Policies with WASL	52
4.2.1 BLP in WASL	52
4.2.2 Biba in WASL	54
4.2.3 Chinese Wall in WASL	57
4.3 Rule Construction for Hybrid Policies	59
4.4 Suitability of WASL for a WSN	60
4.5 Conclusion	63
V. Composition in WASL	66
5.1 Defining Policy Composition	66
5.2 Composition Examples	69
5.2.1 BLP–Biba Composition	70
5.2.2 BLP–CW Composition	70
5.3 Conclusion	72
VI. WASL: The Formal Model	73
6.1 BLP Model Formalized	73
6.1.1 Formalisms of BLP	73
6.1.2 Composition of BLP Systems in WASL	79
6.2 Biba’s Model Formalized	82
6.2.1 Formalisms of Biba	83
6.2.2 Composition of Biba Systems in WASL	86
6.3 CW Model Formalized	88
6.3.1 Formalisms of CW	88
6.3.2 WASL Implementation	91
6.4 Conclusion	93
VII. Conclusion	94
7.1 Contributions	94
7.2 Recommendations for Future Work	95
Bibliography	96

List of Figures

Figure		Page
1	Sample WSN Environment	5
2	System \mathbb{P}	23
3	CW Write Violation	25
4	Statements in WASL	39
5	Types of Terms (Identifiers) in WASL	44
6	Subjects for \mathbb{J}	53
7	Objects for \mathbb{J}	53
8	Structure and Components of \mathbb{J}	53
9	Specification of \mathbb{K}	55
10	\mathbb{K} Compilation Results	55
11	Construction of \mathbb{L}	56
12	Initial Relationships in System \mathbb{P}	57
13	Derived Write Accesses in \mathbb{P}	58
14	System Authorizations for \mathbb{P} After First Compilation	58
15	Additional System Authorizations for \mathbb{P} After Second Compilation	59
16	Unsecure Composition	67
17	Code for \mathbb{P} Subjects' Accesses	71
18	BLP Systems \mathbb{J} and \mathbb{K} Specifications	79
19	BLP System Authorizations Generated by Compilation	80
20	Biba Systems \mathbb{L} and \mathbb{M} Specifications and Generated Authorizations	87
21	CD/COI Structure for System \mathbb{P}	91
22	Object/CD Structure for System \mathbb{P}	91
23	Initial Subject Accesses in System \mathbb{P}	92

List of Tables

Table		Page
1	Query Classifications	11
2	Objects in the BLP Model	16
3	Example System’s Elements Expressed in BLP Model’s Sets	18
4	Sample Policy’s Discretionary Access Matrix	18
5	Sample BLP Policy	19
6	ASL Types and Variables	27
7	ASL Predicates	27
8	Populated ASL Types for Sample Policy A	28
9	Populated ASL Types for Sample Policy B	30
10	Policy-to-Logic Translation	35
11	WASL Grammar	44
12	WASL Relation Term Requirements	46
13	5-Object BLP Policy Compilation Times	62
14	Times for Composition of a 5-Object, 10-Subject System with a 5-Object, X-Subject System	64
15	Times for Composition of a 5-Object, X-Subject System with a 5-Object, 10-Subject System	64
16	Composition Operators, Symbols, and Semantics	69
17	Symbols, Semantics, and WASL Encoding for BLP	74
18	Additional Symbols, Semantics, and WASL Encoding for CW	89

COMPOSABLE DISTRIBUTED ACCESS CONTROL AND INTEGRITY POLICIES FOR QUERY-BASED WIRELESS SENSOR NETWORKS

I. Introduction

The US military is rapidly incorporating wireless technologies into weapon systems to streamline operations. One such example is radio frequency identification (RFID) tags attached to shipped parts to improve the planning, tracking, and management of supplies and equipment. Another is the use of unmanned aerial vehicles (UAVs) to enhance surveillance, communication, and even weapon-delivery options. Wireless sensor networks (WSNs) are a rapidly-changing technology area that, in a sense, bridges between RFIDs and UAVs, providing a potentially-mobile, dynamic network of tiny sensor/communicator devices that can increase the efficiency and effectiveness of the US armed forces. These devices also introduce new vulnerabilities and challenges, however, some of which are described in the following scenarios.

1.1 Scenarios Identifying a Technology Gap

Suppose a mountainous area is a known haven for terrorists. To assist in identifying their specific locations and movements, thousands of microsensors are dispersed in the region. These nodes, or motes, self-organize into a network to support the mission.

While all motes are the same, the network organizes itself such that some fill the role of data provider while others are data integrators. The providers sense events in the environment and forward this data to other motes. The integrators collect this raw data and develop a concise representation of events that have occurred within the sub-region that is in close proximity to that node. The integrators determine how many movements have occurred as well as the origin and destination of each of these movements. Integrators also forward their information to other motes so that eventually the integrators share sub-regional information to build a complete regional representation. A gateway computer administers the network, but even this computer cannot directly access all the motes. Additionally, the gateway is not always available to respond when the network receives a query.

Properly equipped military vehicles that pass near the network are permitted to perform queries to access information in the network. Only certain vehicles' communications and computer systems have the ability (i.e., the proper identity) to modify the network with information such as software updates and to query individual motes regarding the particular data collected.

Suppose that after the WSN has been deployed, several countries are added to the coalition of partners working in the region. To facilitate information sharing and to improve the coalition's

responsiveness to opportunities for action, certain partners are provided user identifiers and authentication protocols that permit them to query the WSN for information. The access protocols are identical for all countries, but each is authorized a unique set of authorizations in the system. The security policy within the WSN is all that requires an update to permit the new accesses.

Over time the need for information sharing with one or more of the coalition partners is likely to change. Further updates of the security policy within the WSN will permit the addition or removal of permissions as appropriate.

This scenario highlights the need for a WSN that can be responsive to changes in users' status. Authentication is assumed and forms the basis of security policy enforcement, but having the ability to quickly and easily update the security policy provides an indispensable level of flexibility for the WSN administrator.

In another scenario, suppose a WSN called the Geo-Weather Sensor Net (GWSN) is deployed by some university across some geographic region. The nodes are equipped with sensors for detecting seismic activity and current weather conditions; some nodes have the responsibility for aggregating data to provide a historical account. Because this network is deployed for research purposes, a variety of research groups at the university collaborate to develop a common interface to the network.

GWSN consists of 5,000 nodes with one notebook computer that provides network management. The notebook computer is not always available and, because the region is accessible to researchers, nodes may be directly queried by properly configured wireless communication devices.

Three groups of researchers require access to GWSN, each group has its own identity but all use the same communication protocols and interfaces. Each group is limited with respect to the information it may retrieve from the network. Members of the geology research group have access only to current and historical seismic data while the meteorology researchers' access is limited to current and historical temperature and humidity data. Computer engineering researchers control the gateway and may access diagnostic information and any current sensor readings.

The university's departments occasionally participate in joint research or academically competitive ventures. These activities require sharing access procedures and codes—authorizations—with groups and individuals outside the existing set of users. New accesses generally follow the groupings previously defined with the addition of geology, meteorology, or computer engineering users. But the new GWSN users have additional limitations that must be enforced.

Messaging protocols in GWSN include a means for authentication; each user has a unique code that is sent with messages to confirm a user's identity (including the user's research group). Encryption is required for each transmission to hide this identity code from eavesdroppers. Changing

code schemes requires significant reprogramming of each mote and reprogramming with any regularity is difficult primarily due to the number of motes as each must be updated individually. The reprogramming task is even more challenging because some motes are normally beyond the range of the gateway.

To provide appropriate accesses for geology researchers from another university, the GWSN access policy is modified to reflect a new geology sub-group. The authorizations permit members of the subgroup to authenticate just as any other geology student, but with a few added limitations as defined by the host university. The new policy is transmitted throughout the network and the new accesses are permitted. These new accesses can be modified or terminated at any time without requiring authentication or encryption protocols.

This example highlights the need for a means to modify a WSN's security policy according to an independent authorization scheme rather than relying on a query language, authentication scheme, and encryption protocol to perform policy-management functions.

1.2 *Background*

The above scenarios identify some of the issues that must be addressed in a secure wireless environment. The proverbial man on the street may also be affected by these issues as wireless devices and associated networks become a part of everyday life. Keys to offices and office buildings use wireless entry authorizations; libraries use small encoded devices that wirelessly report information about books; remote controls for devices such as garage doors, stereos, lights, environmental controls, and other household devices are in nearly every American household; wireless internet connections can be found in homes, businesses, hotels, restaurants, and now encompass entire cities; and the list could go on.

As miniaturized devices with wireless technologies become more complex and pervasive as well as less expensive, there are new opportunities for sensor-dependent solutions. The technologically young WSN is showing promise to meet many of these challenges. As described in the above scenarios, a WSN typically consists of a large number of sensors (possibly hundreds or more) in a relatively small area that broadcast data amongst themselves and may forward data to some gateway linking them to the *outside* world. WSN devices can be deployed densely enough to develop a complete picture of an area and are becoming small enough so as to be indistinguishable from their physical environment, thereby blending into the background.

This research is not limited to the above classical descriptions of WSNs [ZG04; RSZ04], but can be extended to at least two other classes of wireless networks that include sensors: passive networks such those in which RFID devices reside and active networks such as those in which UAVs might

be involved. While WSNs are limited to hundreds of kilobytes of memory and processing power less than ten million of instructions per second (MIPS), RFID tags are typically less capable with several thousand bytes of memory and just enough processing power to, for example, program the device with the intended data. Some RFID tags have no power supplies, relying on the querying agent to provide it via RF signals. UAV's have memory and computational limitations, but at a completely different scale; required communications may exceed 250 Mbps (250×2^{20} bits per second) [DSB04] while powerful computing capabilities are necessary for proper air and ground control and to perform assigned tasks.

To facilitate interactions among WSNs and RFID systems, UAVs, or other wireless systems, this research develops a distributed and composable access control scheme that ensures a request for data is from an authorized consumer. If an unauthorized consumer is given information, confidentiality has been violated. Or suppose access is properly requested and granted for an authorized information source and consumer; can the integrity requirements of both be met? On a larger scale, how do these questions apply within a WSN? These questions implicitly assume there is a policy describing what *authorized* means and what *integrity* is. They also imply a query–response system is used or at least available. While authorization and integrity do not address questions about passive information *stealing* by entities eavesdropping on communications, these threats can be addressed through encryption or information hiding schemes.

Figure 1 illustrates some of the challenges related to these objectives. Consider three systems X, Y, and Z interacting with three wireless networks A, B, and C. Each wireless network has some number of sensors wirelessly linked to some number of other sensors and with a connection to the “outside world” established through some gateway. Each WSN operates with a unique policy that includes both distributed access control and integrity constraints. WSN A operates under access and integrity policy A', WSN B uses policy B', and WSN C applies policy C'. Different computer systems and/or networks (e.g., X, Y, and Z) require access to these wireless networks but must comply with the given policies. Interaction among the WSNs may also be necessary, as indicated by the wireless communication lightning bolt between a sensor under Policy A' and a sensor under Policy B'; policy translation or inherent compatibility is required to allow this type of inter–network communication.

1.3 Purpose

This research has three objectives. First, a language is necessary to represent WSN security policies and provide for their composition; this language must be sufficiently expressive to model a variety of security policies and their compositions. The language developed is called the WSN

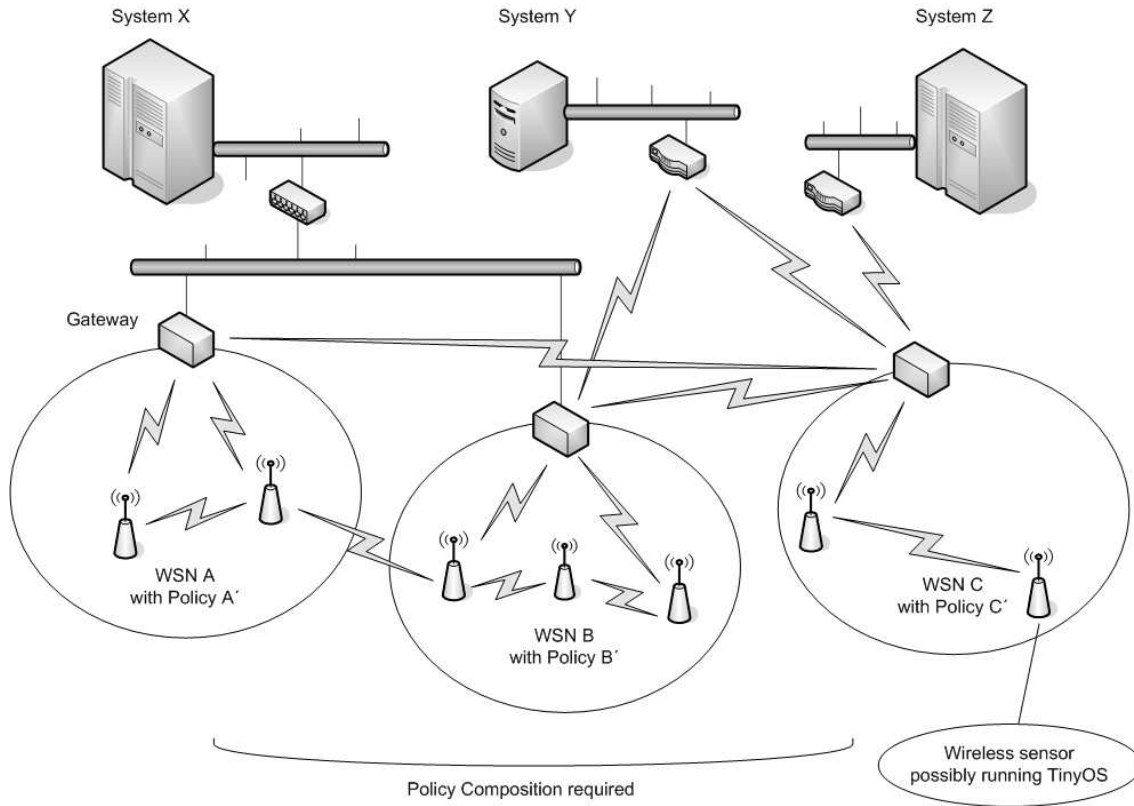


Figure 1: Sample WSN Environment

Authorization Specification Language (WASL). Second, policy composition must be understood and enforceable using this language. Third, the individual and composed policies must be shown to support a specified notion of security as asserted. All three must be defined and structured such that they are relevant to the domain of WSN query-based networks.

A key element in this work is the characteristic of transferring data from one node to another primarily as a response to a query. In the type of network in which any node may receive a query, data transfer permissions are handled either through an arbiter or by the individual node answering the query. The arbiter option is undesirable for a WSN due to severe resource constraints, so the individual node must handle permissions for received queries and the policy-management system must be scaled to account for the capabilities of a mote.

With respect to general WSN security issues, the objectives assume authentication including key management or message protocols; they do not address issues of data freshness. While the security properties of data *confidentiality* and *integrity* are addressed, security of data during transmission is assumed. *Availability* is tangentially-affected in that a security policy identifies the actions permitted (or accessible to) a given user.

The following identifies requirements for the policy system’s language and describes the elements necessary for the composition of policies in this domain. The need for proofs in describing the security of systems implementing the language and composition approach is also presented.

Objective 1: Develop the Language. WASL is the core of this research and it must satisfy several requirements. This language must represent security policies for WSNs and be capable of representing arbitrary policies. While policy specifications may contain a large amount of data, the resulting authorizations of the policy are all that is required at a mote and, therefore, must be suitable for extraction, possible compression, and transmission throughout the WSN.

The language and its associated compiler must support complete delineation of system–permitted authorizations because, again, the motes require this information. Other types of authorizations also must be available as, specifically, mandatory and discretionary authorizations are used in well–known and mature security models.

WASL must additionally provide for the expression of facts regarding relationships among subjects (users) and data objects and between either of these and their respective security levels. These associations are modeled by some researchers as groupings of entities that can also then be referenced by group identifier. To enhance the ability of a security administrator to define and modify a policy, WASL provides rules that make use of stated facts to derive authorizations.

The simultaneous enforcement of multiple partially ordered security levels is a necessary feature for some individual policies and for policy composition. Additionally, role–based security protocols abound and are an element WASL supports.

Objective 2: Define Secure Composition. Maintaining the security of a system after policy composition is vital for a robust solution. But with the many definitions that identify different qualities of a “secure composition,” the attributes needed for query–based WSNs must be selected. Composition must be demonstrated to be feasible for a variety of policy types and be shown to maintain the specified notion of security.

Objective 3: Preserve Security When Using the Language. The policy types used to demonstrate WASL are axiomatic systems, lending themselves to proofs of security. These proofs provide the desired assurance of security. Proofs must be extended to policy specifications expressed in WASL. Similarly, compositions both in theory and as implemented in WASL must be proven to maintain security.

1.4 *Summary*

In response to the aforementioned technology gap, this research accomplishes the following:

- It develops a new language for expressing confidentiality and integrity policies for WSNs and demonstrates its implementation.
- It defines and demonstrates a security-preserving policy composition approach using the new language.
- It uses definitions of security to show that WASL sustains security for individual policies and for composition.

WASL is a means for expressing security policies that can be proven to sustain security according to a given definition. These policies can be composed with others such that the individual policy systems, with subjects permitted and denied certain accesses to particular data within a system, remain undisturbed while additional permissions for inter-system accesses are permitted. These new accesses are shown to be consistent with the defined security definitions as well.

Examples are provided to demonstrate policy specification and composition and the implementation of this new language, complete with a compiler, is also described. Compilation and composition times are discussed to demonstrate the feasibility of the proposal with results identifying both the capabilities of the approach and its utility for implementation within a WSN.

Background information on technical areas related to this research is in Chapter II and includes overviews of WSNs, computer security, and security policy composition. The WASL language is described in detail in Chapter III, including definitions of its grammar and statements and discussions of the compilation and composition of encoded policies. Examples of WASL-encoded policies consistent with Bell-LaPadula's security model, Biba's strict integrity model, and the Chinese Wall model are in Chapter IV, including specifications of hybrid policies. Chapter V defines policy composition and provides examples while Chapter VI supplies the formalization of the policy models presented and shows that the demonstrated policy programs in WASL sustain security as defined by those models. Chapter VII identifies the contributions of this research and follow-on efforts to extend the accomplishments identified herein.

II. Technical Area Review

This chapter presents four important elements of this research. Section 2.1 is an introduction to wireless sensor networks while Section 2.2 presents general information regarding computer security and security policy composition. Section 2.3 provides an overview of methods for expressing and composing security policies. A review of related research is included in Section 2.4.

2.1 *Wireless Sensor Networks*

WSNs [ZG04; RSZ04] sense phenomenon and transmit collected data. They are typically densely deployed to allow short-distance communication among nodes while providing many independent observations within a region. Thus, behavior in the region over time as it relates to the sensed characteristics can be determined. The capabilities and limitations of the sensor within a node fall outside the scope of this research and are not discussed.

The WSNs topology is assumed to be ad hoc and a given node cannot be assumed to be active at any specified time due to device failure, environmental factors, or the transmission protocol itself. Each node may, however, have the ability to identify its location and that of its nearest neighbors. Location may be geographically absolute as determined by the global positioning system (GPS) or some relative measurement as determined by using a factor such as transmission delay or hop distance. This dynamic topology development helps the network adapt to unpredictable environments when a given node is not responsive. Additionally, a gateway device provides network administration and a link between the WSN and an external network. This allows WSN nodes to perform sensing and communications within the WSN independent of external network requirements.

The rest of this chapter discusses the WSN's limitations in Section 2.1.1, communications in Section 2.1.2, and the nature of a query-based network in Section 2.1.3. A brief overview of WSN security issues is in Section 2.1.4 with the description of an operating system designed for WSNs following in Section 2.1.5.

2.1.1 Limitations. Most limitations inherent to the devices in a WSN can be linked to their physical size. For example, power, computational capacity, and communication capabilities are ultimately all constrained because the the nodes are very small devices.

The power supply of each unit is constrained to a battery or other small power source that supports all the sensing, computation, and communication activities of the device. Communications is one of the most power-intensive operation performed by the device [PSSC04], placing an upper limit on the lifetime of any given node. WSNs are designed to withstand the periodic loss of nodes throughout the network, in part, because power failure is a common cause of WSN device failure. Any reduction of communications may extend the lifetime of a node through power savings.

The memory and computational capacity of a mote are also severely limited because of size and power. Complex computational algorithms and manipulations of large datasets are untenable for a WSN node.

An example of the typical computation and communication capabilities and limitations is the MICA2 Dot Mote [Cro07]. The MICA2 Mote weighs 18 g and operates on two “AA” batteries. It hosts the TinyOS operating system on an ATmega128L 8-bit processor running at 8 MHz with 4 KB RAM and 128 KB flash RAM for code. These specifications mean most tasks, including policy management, reside in code that is on the order of hundreds of lines of code in length.

The above limitations on node power, memory, computations, and, by implication, communications significantly constrain the capabilities of individual nodes. Just as for any other task handled by the node, a security policy enforcement mechanism must be designed to use resources sparingly if this capability is to be implemented in a node.

2.1.2 Communication. Radio links are the most common communication method among nodes in a WSN. Each node communicates with its *close* neighbors while communication to *far* nodes is accomplished only after the message has been transmitted through a series of one or more hops from node to node. Every node within range of the transmitting node is a potential receiver of the transmitted information.

Similar to other types of networks, WSNs layer communication protocols to effectively move data from one node to another [SOBAC04].

- The application layer provides a means of identifying nodes by location or capability and prepares the message for transfer.
- Transport layer mechanisms add error control to the data and regulate communication flow while preparing the message for routing.
- Network layer protocols perform the routing within the WSN.
- The data link layer includes medium access control (MAC—providing the infrastructure for multi-hop network management and for ensuring fair sharing of communication resources within the WSN), error control, and data frame detection/data stream multiplexing.
- The physical layer transmits or receives data streams.

Policy mechanisms are implemented in the data link layer where they function as a filter to prevent unauthorized communications. Policy enforcement mechanisms may also be in the application layer to permit certain actions requested by a querying agent while denying others. Whatever

the final implementation, policy enforcement must prevent transmissions in response to queries when the querying agent is not authorized per the established policy.

In the process of carrying out the responsibilities of the network, nodes must communicate with each other. Communications can be event-driven or query-driven, the strengths of each explored in [BE02]. Consider an event-driven WSN with the task of tracking objects through a WSN field. A node must sense pertinent events, process them appropriately, and transmit coordinates, timing, and other information to its neighbors. The event of sensing a moving object serves as the trigger for computation and communication.

WSN activity may alternatively be in response to a query, perhaps from one mote to another, but possibly from an external entity. One could define a query as simply another type of event. This query-driven paradigm is explored next.

2.1.3 Query-Based Network. There are three primary categories of queries in a sensor network [EN03]: single source queries require only one node's information in response, non-aggregate queries require knowledge about multiple sensors but not necessarily information from multiple sensors, and aggregate queries require information from multiple sensors consolidated into a single response. Examples of the three query types are, respectively: Did node X sense event E? Which nodes could have sensed event E? What was the average reading of all sensors when event E occurred? A query from any of these categories may be presented to a node at any time, particularly if the query is coming from an entity outside the network.

Another way to partition queries, independent of the aforementioned categories, is by considering the time frame of the desired answer [BGS00]. Historical queries look to the past, snapshot queries consider the current state of the network, and long-running queries ask a question now that may return many answers in the future or a single answer much later than the query's submission.

An alternative view of queries considers them in four classes [CKP03]: entity-based or value-based and aggregate or non-aggregate. A query labeled as an aggregate requires some level of inter-node communications to determine the answer while a non-aggregate requires a single node's data. Orthogonal to the aggregate and non-aggregate types are the value-based versus entity-based categories. The former returns a single answer whereas the latter returns sets of values.

The query categorization method selected depends on the application, but these varied descriptions, summarized in Table 1, point out some important concepts to consider for a query-based network. When a query is received by the network, a node might require answers to the following questions to determine the appropriate response, with an emphasis on security (security is discussed in Sections 2.1.4 and 2.2).

Table 1: Query Classifications

Distinguishing Factor(s)	Categories
Nodes required to answer a query	Single node multiple nodes
Type of data requested	Data from the nodes, data about the nodes
Time frame of the desired answer	Historical current snapshot periodic, future
Data type and scope	Entity-based non-aggregate value-based non-aggregate, entity-based aggregate value-based aggregate

- What is the protection (or perhaps integrity) level of the querying agent?
- How many nodes' data are required by the query?
- What are the protection (integrity) levels of each of the affected nodes?
- What is the protection (integrity) level of the answer to the query?
- Does the query require retransmission to arrive at the node with the answer? Are node privileges such that retransmissions will not fail at some point due to security?
- Does the answer traverse a number of nodes to arrive at the destination? Is the answer visible to intermediate nodes and/or are intermediate nodes authorized to *see* the answer?

The purpose of each query is to seek an answer regarding events that occur within the WSN [BE02]. The policy controlling reception of and responses to queries may need to specify whether a receiving node should respond to queries or which single node with the answer to the query should respond. The answer may depend on whether the query originates from within or from outside the network.

Security issues are key challenges for establishing communications among nodes and the queries presented to them. Future WSNs may permit a node to respond directly to a query from an extra-WSN node and, if so, security protections will have to account for these nodes as well as all the intra-WSN nodes. The next sections discuss the issues and protocols that address those issues.

2.1.4 Security Issues. WSN characteristics lead to obvious and perhaps not-so-obvious vulnerabilities [CP03; AUJP04]. The following are a few of the many areas, some of which are just now beginning to be investigated.

Since WSN nodes might reside in an area not physically controlled by the user, a malicious user might locate a sensor and alter its performance. Such hijacking could result in improper data being transmitted or could lead to unauthorized access to information in the WSN. The adversary might simply add an unauthorized node to the network for various active or passive malicious attacks.

Most WSNs broadcast data. Simply monitoring data broadcasts may be all an adversary needs to do to capture the state of the WSN or the information being transmitted unless encryption is used. With encryption the task becomes more difficult for the malicious user, but significant resources must be expended to perform encryption at each node.

Data that resides in a given node or collectively on the network effectively has no protection at all if an adversary can capture it with a simple query of the network. Such a query may be made indirectly, as the information gathered by the WSN is transmitted to authorized users directly or via the connection of the gateway to the Internet or some other generally-accessible network. If an adversary understands enough about the network's behavior, traffic patterns might reveal important information about the network. Therefore, traffic analysis is also a security issue.

Other issues include those identified in the following questions. How does the network deal with a malicious node that either fails to forward information or forwards incorrect information? How is the threat of a compromised WSN-wide encryption key identified and mitigated? How do the nodes maintain all the cryptographic information required if point-to-point keys are employed?

Many questions are answered by establishing protocols for data security. One approach separates the concerns of data security from broadcast authentication. Data security has four requirements [PSW⁺01] that closely parallel the generic computer security issues in Section 2.2, but the definitions for confidentiality and integrity are much weaker below.

Data Confidentiality protocols typically employ some level of encryption to provide secure communication of data to approved nodes. This includes *semantic security* by which an eavesdropper cannot discern anything about or recognize a message even if the same message is transmitted multiple times.

Data Authentication ensures a message comes from an authorized source and the source is the originator of that message. More detail on authentication follows.

Data Integrity ensures data is unaltered during transmission and verified with data authentication.

Data Freshness describes the ordering and currency of data. *Strong freshness* is a total ordering of messages as well as the time delay of transmission and *weak freshness* ensures only proper ordering.

Symmetric and asymmetric approaches to data authentication have been proposed [PSW⁺01; Lam79; GR97; Roh99; PCST01]. In symmetric authentication the sender and receiver share a key that enables the receiver to verify the sender's identity. Asymmetric authentication requires the transmission of a digital signature of significant size. A conventional symmetric authorization key sent with each message is cost prohibitive for a sensor network, much less the asymmetric signature;

each approach consumes too much bandwidth, memory, or computational power for a typical sensor node [PSW⁺01]. While authorizing manipulation of network data is central to discussions below, prior authentication is presumed herein.

This research has developed a solution to the security questions that arise after a node has received a request from an authenticated source. It answers the question, “What are the requestor’s privileges based on security classification(s) and access rules possibly related to those classifications?”

Discussions about authentication focus on identifying users are who they claim to be; but that is only part of security. Other components include the areas of confidentiality and integrity that are investigated in Section 2.2.

2.1.5 TinyOS. A specialized operating system (OS) called *TinyOS* has been developed at the University of California, Berkeley [HSW⁺00], to support a family of WSN motes. It was designed specifically to meet the power, memory, and processing limitations of small, untethered sensor nodes.

TinyOS has a layered architecture, where *down* refers to those layers closer to the hardware and *up* to those further away. It is an event-driven architecture where three types of activities are performed: tasks, commands, and events. In general, events signal upward while commands are issued downward with tasks providing the bulk of required computations. This layered architecture supports component-based construction of the desired system.

An extension to the C programming language, called *nesC* provides a means of building components that meet the unique requirements of TinyOS [GLvB⁺03; GLCB03] and TinyOS applications. While conforming to the TinyOS concept of components, *nesC* does not support dynamic memory allocation, but facilitates *whole-program analysis*. This results in simpler and more accurate safety analysis, provides optimization of the entire program (as opposed to single applications), and entirely eliminates the overhead of memory allocation schemes. Each component created for TinyOS is composed of event handlers, commands, and/or tasks.

Tasks are data manipulation routines that constitute the primary work of the node. Each is completed in the order it is placed on the task scheduler’s list. They signal higher level events or call lower level commands, and each must be completed prior to beginning another task. Tasks can be interrupted by events.

Commands are calls that may post tasks on the task scheduler or invoke lower level commands. Each returns a “success” or “failure” status to the caller without significant delay, so it cannot wait long for subordinate commands to terminate.

The lowest level *events* are essentially hardware interrupts that are handled immediately by the processor. Events are permitted to call higher-level events or commands and may result in the addition of tasks to the schedule. They can be interrupted by higher-priority events.

Each of these three activities is limited both in execution and in the actions that can be invoked. These limitations provide a great deal of protection against long running tasks, but cannot eliminate them.

TOSSIM is a useful testing environment when nodes are not available. It simulates TinyOS at the bit level [LL03]. Every interrupt is simulated, every line of application code is run, and interrupt timing is precise when actual TinyOS code is run. It is limited, however, in that each piece of code runs sequentially with the appearance of instantaneous execution. That is to say, it cannot detect faults that might occur in real nodes due to colliding interrupts or task preemption during execution.

Among the various additions to TinyOS is a security suite called *TinySec* [KSW04]. The goals of TinySec include improvements in access control, integrity, and confidentiality. At its core, TinySec uses a shared key to encrypt and decrypt each message sent and received. Failure to use the same key results in message rejection. It can also add sequencing or a time-related key within a symmetric authorization protocol to provide an additional low-overhead data security mechanism. Such a mechanism may prevent eavesdroppers from understanding or recognizing broadcast information that is repeatedly transmitted [PSW⁺01]. This mechanism contributes to security during transmission and is a primary component of an authorization scheme. It does not, however, address the requirements of policy implementation or policy composition.

2.2 *Computer Security*

This section provides fundamental information in computer security, particularly as it relates to access control and integrity, terms that are defined below. Discussed herein are central principles of security along with models reflecting some of the various aspects of security. Formalisms used for evaluating security principles are introduced along with the models, when applicable.

To whom should a system allow access and upon what criteria is this access based? Once access has been granted, what privileges should be permitted? How are these permissions expressed? How are permissions verified? These questions and the answers to each define computer security for a given system.

The context of an access request (i.e., the history of accesses) may matter a great deal. Additionally, the role of a subject, discussed as the “user” above and defined herein as an entity that can act with respect to the protection system, may be the primary consideration regarding access.

The rights of a subject may change as the actor moves into and out of particular roles. Access may alternatively be granted solely based upon a subject's identity with each subject possessing specific capabilities because of some unique characteristic.

Before presenting details of specific security principles, the computer security foundation is laid in Section 2.2.1. The most relevant of the overarching principles are confidentiality and integrity, respectively addressed in Sections 2.2.2 and 2.2.3. The *Chinese Wall* security model in Section 2.2.4 illustrates how some of the fundamental principles are combined in a security system.

2.2.1 Fundamentals. Computer security addresses three primary components [Bis03]: confidentiality, integrity, and availability. Confidentiality and integrity are discussed in more depth below, as they are central to the purposes of this research.

Availability means that authorized subjects can access data, information, or resources on demand. The failure of a system to provide the desired availability may result from natural (e.g., hardware failing) or hostile causes (e.g., denial of service (DOS) attacks). The inability of the system to perform as intended is considered a security issue, particularly as a result of a hostile agent's actions.

DOS attacks come in many forms. Flooding a communication channel or submitting an overwhelming number of data requests are two examples of this type of attack. For WSNs another plausible DOS attack includes any actions that intentionally cause excessive transmissions over a short period of time, resulting in power consumption that shortens the life span of the nodes. Thus, the prevention of unnecessary communications is a primary goal of a WSN security system.

The reliability of a system also plays a role in availability. Authorized users need some level of confidence that a system will provide the requested response within its design limitations. Security policies help ensure this and can also help prevent DOS attacks by identifying permitted accesses and eliminating any system response for denied accesses.

An organization having an understanding of these three computer security elements can establish computer security requirements and identify security policies enforced by security mechanisms. Security mechanisms are the means of achieving the requirements by identifying the "method[s], tool[s], or procedure[s] for enforcing a security policy" [Bis03].

Encryption and information hiding are a primary focus of many researchers today, particularly as relates to WSNs. These mechanisms address many security concerns in the area of integrity and access control and contribute significantly to the security of networks. However, these efforts do not address policy management.

Conversely, this research is not concerned with the mechanisms, but rather with security policies. The interest here is specifically in policies that specify the confidentiality and integrity requirements of a system. It considers statements of what actions are, or are not, permitted. Security policies are discussed in more detail in Section 2.3, but Sections 2.2.2 and 2.2.3 consider confidentiality and integrity in the context of security policies.

2.2.2 Confidentiality. At its core, confidentiality means individuals (or processes) cannot access information unless they are authorized to do so. Assurance of confidentiality implies that secrets remain secrets—protected data is hidden from all but the intended audience.

For this confidentiality discussion, consider a system with two security levels, **S** and **U**. The system contains subjects and objects associated with each of these levels. Level **S**, and thus every entity associated with this security level, has a higher confidentiality (or classification) level than level **U** (and associated entities).

2.2.2.1 Bell-La Padula Model. Information flow analysis can describe, in part, the confidentiality of a system. The confidentiality requirement from the Bell-LaPadula Model (BLP) [BL75] dictates that reads and writes by subjects of objects (say, files in this example) are permitted except that subjects at level **U** may not read from level **S** files (the simple security condition) and subjects at level **S** may not write level **U** files (the *-property). Because the presumed ways for information to flow from one object to another is by reading or writing, this scheme preserves confidentiality by specifying that information categorized as **S** shall not flow into a **U** file (or to a **U** subject) by either method.

A formal representation of the BLP model is presented here with slight notational modifications [Bis03]. The system consists of the sets of objects identified in Table 2.

Table 2: Objects in the BLP Model

Set	Description
A	the set of all actions
F	the set of subject and object clearances per state
H	the set of object hierarchies per state
M	the set of sets of permitted accesses per state as contained in a matrix representation
O	the set of all objects
S	the set of all subjects
V	the set of all system states

A state $v \in V$ can be represented as (b, m, f) where $b \in P(S \times O \times A)$ (using $P(x)$ to indicate the powerset of x) are the rights permitted per the security definitions, and $m \in M$. Hierarchies $h \in H$ can be included in a state v , but for a WSN, the hierarchies do not change over time and are therefore omitted. m are discretionary rights—rights assigned to a subject by a security administrator. The functions f_S , f_C , and f_O identify a subject’s maximum and current security levels and an object’s security level, respectively. $f \in F$ can be the triple (f_S, f_O, f_C) , but in a WSN the subject is not expected to change security levels and, therefore, the discussion below considers f_S as the sole security level of the subject. Thus, $f \in F$ is the pair (f_S, f_C) . If a subject $s \in S$ is permitted action $a \in A$ on object $o \in O$, then $a \in m[s, o]$ where $m \in M$.

The domination function, dom , defined below, is true when one entity has an equal or higher security level than another (i.e., $f_S(s_1) dom f_O(o_1)$ means that subject s_1 ’s current security level is greater than or equal to the level of o_1).

Definition 2.2.1. *The security level l_1 dominates the security level l_2 if and only if $l_1 \geq l_2$ and is written $l_1 dom l_2$.*

Let $a \in A$ where $A = \{\underline{r}, \underline{w}\}$, for the respective actions read and write. The presentation in [BL75] identifies two other actions, namely, execute (meaning an action that includes neither read nor write) and the combination read–write (where information flows both toward and away from the actor in the same action). Analyzing the absence of data flow (e.g., the action execute) adds nothing to this research and is, therefore, omitted. On the other hand, an action in a WSN is expected to be either read or write and, thus, the combined read–write action is also omitted.

The *simple security condition* in Definition 2.2.2 below identifies the first mandatory access requirement for a system to be considered secure: it cannot permit a subject with a given security level to read an object with a higher security level. This is commonly stated, “reads up are not allowed.”

Definition 2.2.2. *$(s, o, a) \in S \times O \times A$ satisfies the simple security condition relative to f if and only if one of the following holds:*

- a. $a = \underline{w}$
- b. $a = \underline{r}$ and $f_S(s) dom f_O(o)$

The simple security condition identifies particular accesses that satisfy the simple security condition, but a state $v \in (b, m, f)$ satisfies the simple security condition if $ssc rel f$ is satisfied by every element of b . If every state satisfies the simple security condition then the system satisfies the simple security condition.

The $*$ -property, defined next, addresses access requirements for a given state, where $b(s : a_1, \dots, a_n)$ is the set of objects to which s has $\{a_1, \dots, a_n\}$ rights (i.e., $b(s : a_1, \dots, a_n) = \{o \mid o \in O \wedge [(s, o, a_1) \in b \wedge \dots \wedge (s, o, a_n) \in b]\}$). This definition identifies the additional requirement that “writes down are not allowed.”

Definition 2.2.3. *A state (b, m, f) satisfies the $*$ -property if and only if, for each $s \in S$, the following holds:*

- a. $b(s : \underline{w}) \neq \emptyset \Rightarrow [\forall o \in b(s : \underline{w})[f_O(o) \text{ dom } f_S(s)]]$
- c. $b(s : \underline{r}) \neq \emptyset \Rightarrow [\forall o \in b(s : \underline{r})[f_S(s) \text{ dom } f_O(o)]]$

The third requirement for a secure system is in Definition 2.2.4. Permissible system accesses must all appear in the discretionary rights of the access control matrix.

Definition 2.2.4. *A state (b, m, f) satisfies the discretionary security property (ds-property) if and only if, for each triple $(s, o, a) \in b$, $a \in m[s, o]$.*

Finally, system security is defined in

Definition 2.2.5. *A system is secure if it simultaneously satisfies the simple security condition, the $*$ -property, and the discretionary security property.*

Consider subjects s_s and s_u and files (objects) o_s and o_u and the other populated BLP model sets shown in Table 3. An example of a discretionary access matrix in Table 4 complements the model. The arrow \rightarrow indicates a mapping from the entity on the left to the entity on the right.

Table 3: Example System’s Elements Expressed in BLP Model’s Sets

Set	Contents
S	$\{s_s, s_u\}$
O	$\{o_s, o_u\}$
A	$\{\underline{r}, \underline{w}\}$
V	$\{v\}$
M	(see Table 4)
F	for state v , for every function f : $f_S(s_s) = f_S(s_u) = \mathbf{S}$, $f_O(o_s) = \mathbf{S}$, $f_S(s_u) = f_S(s_u) = \mathbf{U}$, $f_O(o_u) = \mathbf{U}$
H	$\{S \rightarrow \{U\}, U \rightarrow \{\emptyset\}\}$

Table 4: Example System’s Discretionary Access Matrix

Subj	Obj	
	o_s	o_u
s_s	$\underline{r}, \underline{w}$	\underline{r}
s_u	\underline{w}	\underline{r}

Table 5 identifies the possible actions by the subjects in the system and identifies whether or not each action is allowed and why, illustrating the effectiveness of the BLP model in capturing confidentiality for the example system.

Table 5: Results for BLP in Example System

Question	Result	Reason if Disallowed
Can s_u perform \underline{r} on o_u	Allowed	
Can s_u perform \underline{w} on o_u	Disallowed	Def 2.2.4, $\underline{w} \notin m[s_u, o_u]$
Can s_s perform \underline{r} on o_u	Allowed	
Can s_s perform \underline{w} on o_u	Disallowed	Def 2.2.3, $\neg f_O(o_u) \text{ dom } f_S(s_s)$
Can s_u perform \underline{r} on o_s	Disallowed	Def 2.2.2, $\neg f_S(s_u) \text{ dom } f_O(o_s)$
Can s_u perform \underline{w} on o_s	Allowed	
Can s_s perform \underline{r} on o_s	Allowed	
Can s_s perform \underline{w} on o_s	Allowed	

The BLP model defines confidentiality in an axiomatic system. The formal definitions and representations provide the means for evaluation of various systems with respect to the model. This depiction is not, however, a comprehensive model of confidentiality.

2.2.2.2 Other Confidentiality Models. Suppose the above system implements tasks on a first-come, first-served basis and employs a shared memory space. With this architecture there are new confidentiality concerns. Can a low-side subject (i.e., at security level **U**) learn anything about a high-side file (i.e., at security level **S**) by observing memory usage? Certainly a low-side subject can identify something about the high-side if it can directly or indirectly monitor processor availability. It may also be that high-side inputs could alter low-side outputs.

These concerns are captured in the concepts of *non-deducibility* and *noninterference*. A system is non-deducibly secure if a subject with a lower confidentiality level cannot determine anything about the inputs of subjects with a higher confidentiality level [Sut86]. Noninterference (defined for certain deterministic systems) is slightly stronger and addresses security from a different perspective, namely, whether a distinct group of subjects' inputs into the system will impact the output of subjects from another group [GM82; McC90].

Consider the following example. Non-deducibility is violated if a low-side subject can read memory space that has been written by a high-side subject; even if the written data is encrypted non-deducibility would hold because subjects at level **U** are not allowed to deduce anything about the inputs by subjects at level **S**. Even with encryption, the low-side subjects might be able to determine information about the input data by recognizing the the changing size of of available memory space or the inability to make immediate use of the processor. In this latter case the information transferred is not the high-side data itself, but characteristics of the data.

Noninterference is manifest in this system if any sequence of low-side inputs results in different low-side outputs, depending on the particular sequence of high-side events (i.e., high-side events affect low-side functionality). In this sense it is stronger than deducibility: a low-side subject's ability to identify that a high-side event has occurred violates noninterference while it doesn't necessarily violate non-deducibility.

To contrast the BLP model with the properties identified here, BLP considers information transfer in a direct sense—a subject must read or write to an object for information to flow between the two entities. Non-deducibility and noninterference capture potential disclosures via indirect means. Satisfying the BLP criteria could conceivably be done through software modifications while meeting non-deducibility and noninterference requirements likely requires a combined hardware–software solution.

Confidentiality is a central element of access control. It determines who has access to resources and who is prevented from performing certain actions or from accessing particular data; it identifies which subjects have what rights to what data. Confidentiality addresses whether or not data has been (or could be) compromised. Integrity complements this aspect of computer security and is described next.

2.2.3 Integrity. A system with integrity provides trustworthy data and reliable communications; this property guarantees the origin of data (called *authentication*, cf. Section 2.1.4) and prevents the improper manipulation of data [Bis03]. In one sense, integrity is addressed by mechanisms and policies similar to confidentiality. Data integrity is maintained if only those authorized to handle it have done so.

With the word *integrity* comes the implication of trust. Consider an example in which subjects and objects belong to one of two levels of trust, trusted (**H**) and untrusted (**L**). If a subject, s_h , at integrity level **H**, writes to an object o_l , at level **L**, there is no diminishing of trust— o_l is no less trustworthy than before. If an untrusted subject, s_l , writes to a trusted object, o_h , however, the newly written information (at integrity level **L**) puts untrusted information in o_h thus causing it to become an untrusted object.

This is, in part, what the Low Water Mark Policy was designed to prevent [Bis03]. The Low Water Mark Policy allows trusted subjects to write to less trusted objects only if the integrity level of the object is dominated by (i.e., is less than or equal to) the integrity level of the subject. It does, however, allow a subject to read from an object with a lower integrity level, but the subject's integrity level is then reduced to that of the object read. The rules for the Low Water Mark Policy are

1. $s \in S$ can write to $o \in O$ if and only if $i(o) \leq i(s)$.
2. If $s \in S$ reads $o \in O$, then $i'(s) = \min(i(s), i(o))$, where $i'(s)$ is the subject's integrity level after the read.
3. $s_1 \in S$ can execute $s_2 \in S$ if and only if $i(s_2) \leq i(s_1)$.

where function $i(x)$ returns the integrity level of x and $\min(i(s), i(o))$ returns the lower of the integrity levels $i(s)$ and $i(o)$. *Execution* models inter-process communications, such as procedure calls.

The primary weakness of this scheme is that a subject's integrity level is non-increasing leading to the possibility that subjects may eventually lose all permissions to read trusted (higher integrity) objects. A change to rule 2 remedies this situation and is called the Ring Policy [Bis03]:

2. A subject may read any object, regardless of integrity levels.

In this scheme subjects have the option to access objects in the various integrity groups and the integrity of objects is preserved, although only with respect to direct modification. This implies the subjects are designed to properly handle the various levels of trust. Any formal representation of the enforcement of this security policy implies the subject must contain security enforcement structures within itself. If this is not the case, the scheme will quickly fail to ensure any meaningful notion of integrity.

Biba's Strict Integrity Policy model [Bib77] takes a more restrictive stance with respect to integrity and is the dual of the BLP model. It has two axioms that are the inverses of the simple security condition and the *-property to prevent direct and indirect data modifications that might be harmful to the trustworthiness of the information in the system. The simple security condition prevents higher integrity subjects from *reading* lower-integrity objects while the *-property prevents lower-integrity subjects from *writing* to higher-integrity objects. Together these preserve Biba's notion of trustworthiness.

The following contrasts Biba's terminology and that of BLP:

- An *integrity level* in Biba corresponds to a security *level* in BLP.
- The *leq* relation in Biba's model is the inverse of BLP's *dom* relation.
- *Observe* corresponds to *read* and *modify* to *write*.

Building on both the BLP and Biba's Models, Lipner [Bis03] developed a scheme where both confidentiality and integrity are simultaneously enforced. While trust is a key element of this model, *separation of function* and *separation of duty* are also introduced to facilitate policy requirement of

complex systems. Lipner’s “Integrity Matrix Model” specifies two security levels and five categories of data. Based on the roles of the subjects (e.g., developmental programmers, auditors), the privileges associated with them (e.g., low security developmental code, production code), and the limitations imposed on them (e.g., a developmental programmer should not be permitted to modify production code), the effectiveness of implementing confidentiality and integrity principles in a realistic system was demonstrated.

Integrity policies can cover a wide range of security requirements. Most applicable are those aspects related to access control that address questions like

1. Given a subject is granted a certain level of trust, what actions does the policy permit?
2. Given an integrity-preserving policy, how does one ensure that when it is composed with another integrity policy, the original policy remains intact?

The following model is neither simply an integrity policy nor a confidentiality policy, but enforces principles from both. While Lipner’s model is relevant to real-world scenarios, the Chinese Wall model provides a slightly more complex model with formalisms.

2.2.4 Chinese Wall Model. The *Chinese Wall* (CW) addresses commercial concerns regarding access to data from competing companies. This model has been recognized as useful for preserving integrity requirements in practice ([BN89] notes that the CW model has been used as **the** required model for the United Kingdom Stock Exchange). This model provides access for subjects to objects while preserving specific requirements to prevent conflicts of interest.

In some business settings it is vital that individuals with access to certain information be prevented from accessing other information to avoid conflicts of interest with respect to the data in the system. The discussion that follows is based on the presentation in [BN89].

Suppose there is the set of objects, O , and a set of subjects, S . In the example of Figure 2 (further elaborated below) elements of O are distributed among conflict of interest (COI) classes ($COI1$ and $COI2$), that contain datasets (CDs) that should not be accessible to a given subject when another CD in the COI class has been accessed by the same subject. Let the CDs in $COI1$ be $CD1$ and $CD2$ while $CD3$ and $CD4$ are in $COI2$. One other CD, *sanitized*, is labeled $CDSan$ and resides in its own COI class, $COISan$.

A subject s begins with the ability to access (or *read*) data in all datasets. Once the subject has accessed a data item, say, $D3$, from $CD2$, the data in the other dataset ($CD1$) within the same COI class ($COI1$) becomes inaccessible to s . Objects in the *sanitized* CD ($CDSan$) can be accessed without regard to previous accesses because it resides in its own COI class.

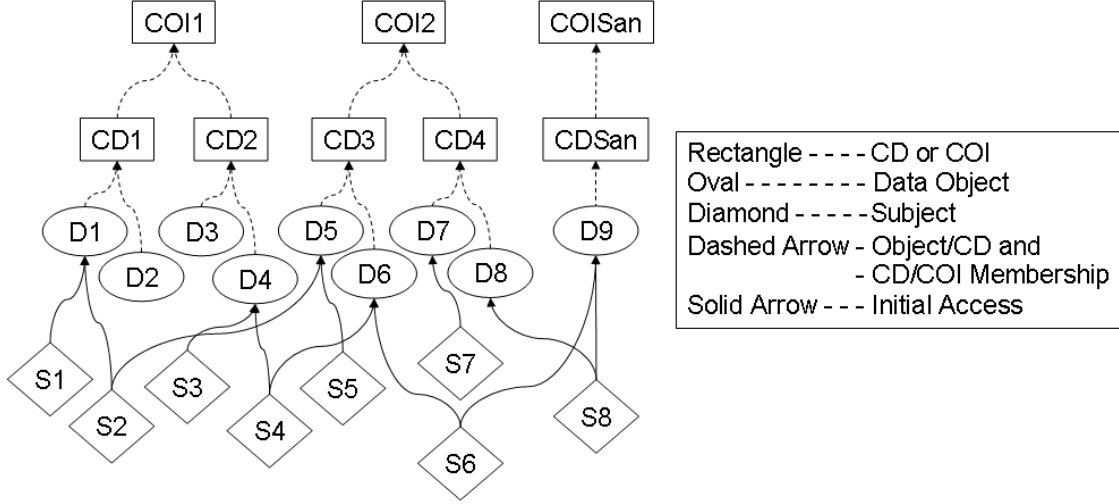


Figure 2: System \mathbb{P}

Write permissions are restricted so that indirect violations of the security principles are not possible (e.g., reading via a “mailbox”). A subject may only write to a data object if both *read* authorization to the object is already granted and there are no *read* authorizations for data objects in any other CD with the exception of *CDSan*. Otherwise, the subject might divulge restricted information to other subjects who lack permissions to obtain that information.

To begin formalizing the model let L be the set of security labels (x, y) with one label per object, where x and y represent the COI class and CD, respectively. Functions $X(o)$ and $Y(o)$ return the x and y components, respectively, for the label associated with object o . Thus, x_1 is the COI and y_1 is the CD for o_1 for $X(o_1)$ and $Y(o_1)$ respectively. A request r is formatted $r(h, j)$ where subject s_h is requesting to read object o_j . The sanitized COI and CD are denoted x_o and y_o , respectively.

The following axioms define fundamental properties of the CW security system. The first states that objects in the same CD also reside in the same COI class. The second is the converse—if two objects are in different COI classes, they must be in different datasets.

Axiom 2.2.1. $y_1 = y_2 \Rightarrow x_1 = x_2$

Axiom 2.2.2. $x_1 \neq x_2 \Rightarrow y_1 \neq y_2$

\underline{N} is an $\|S\| \times \|O\|$ matrix (where $\|S\|$ denotes the cardinality of S) that tracks which subjects have accessed which objects, using boolean entries in the matrix. $\underline{N}(h, j)$ identifies whether subject s_h has accessed object o_j (i.e., the matrix entries are either *true* or *false*).

Definition 2.2.6. \underline{N} is a boolean matrix with elements $\underline{N}(h, j)$ corresponding to members of $S \times O$. Entries are true if s_h has or has had access to o_j . When a new access is granted to or performed by s_k for object o_l , \underline{N} is replaced by \underline{N}' , a matrix identical to \underline{N} except that the entry for $\underline{N}(k, l)$ is true.

Read access to an object, l , is granted if and only if the subject, k , has never accessed another object in the COI class or if the object is in the same dataset as previously accessed objects.

Axiom 2.2.3. $r(k, l) \rightarrow \underline{y} \iff \forall \underline{N}(k, j) = \text{true}, ((x_j \neq x_l) \vee (y_j = y_l))$

The initially secure state is defined and, given this state, the first request for access is permitted.

Axiom 2.2.4. $\forall (h, j)(\underline{N}(h, j) = \text{false})$ is an initially secure state.

Axiom 2.2.5. $\forall (h, j)(\underline{N}(h, j) = \text{false}) \Rightarrow \forall (h, l)(r(h, l) \rightarrow \underline{y})$

Sanitized data has unique properties:

Definition 2.2.7. For any object o_j

- $y_j = y_o \Rightarrow o_j$ contains sanitized information
- $y_j \neq y_o \Rightarrow o_j$ contains unsanitized information

The next three theorems establish the basis for the model.

Theorem 2.2.1. Once a subject has accessed an object, the only other objects accessible by that subject lie within the same company dataset or within a different COI class.

Theorem 2.2.2. A subject can at most have access to one company dataset in each COI class.

Theorem 2.2.3. If for some COI class X there are n company datasets, then the minimum number of subjects which will allow every object to be accessed by at least one subject is n .

Even given the above structure there is the possibility of inappropriate indirect transfer of information. Suppose there are two COI classes X_a and X_b with datasets D_{ax} and D_{ay} from class X_a and D_{bx} from class X_b as presented in Figure 3. Further suppose subject S_u has access to D_{ax} and D_{bx} while subject S_v has access to D_{ay} and D_{bx} . S_u is permitted to read from D_{ax} and write it to D_{bx} . This indirectly provides the possibility of data transfer from D_{ax} to S_v , effectively violating COI rules for class X_a via S_v .

The following theorem addresses this possibility using the concept of *sanitized data*. Sanitized data removes any sensitive information from data that is written to a different dataset. The enforcement of this protection comes from a final theorem:

Theorem 2.2.4. No object containing unsanitized information can be read from a company dataset that is different than the one for which write access is requested.

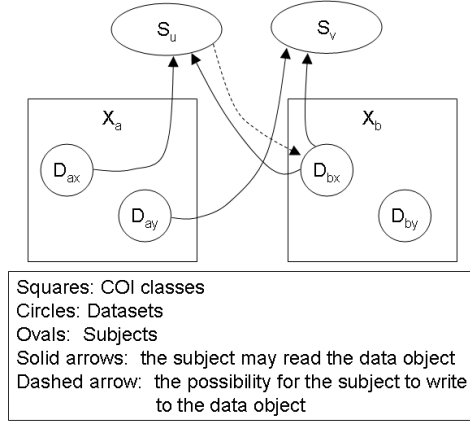


Figure 3: CW Write Violation

The CW model represents neither a confidentiality nor an integrity security policy, but has elements of both. With respect to confidentiality, CW prevents unauthorized subjects from accessing data. On the other hand, a conflict of interest would be a breach of its notion of integrity and CW prevents such breaches. CW, as is BLP, is an axiomatic system with which security assertions and mathematical comparisons can be made to analyze systems employing its principles.

The Chinese Wall, along with the other models above, demonstrates what is perhaps a more important element of formalized modeling: that the authorization of a subject to perform some action on an object is unambiguous. As has been observed above, work with WSNs requires that components implemented in the nodes be small with respect to memory and computation requirements. Clear delineation of subject's rights should reduce computation requirements and, as long as the representation of subjects and the actions each may perform on the various objects is compressed sufficiently, memory requirements should be modest as well.

2.3 Security Policy Representation and Composition

Several elements are required to compose security policies in practice. The policies must be expressed appropriately, a framework must exist to compose those policies, and the resulting policy information must be transformed into representations that can be expressed logically for computer programs. These areas are addressed successively in Section 2.3.2, 2.3.3, and 2.3.4. Section 2.3.1 first introduces some basic concepts.

2.3.1 Introduction to Security Policy Composition. Research into security policy composition uses the specific security properties discussed in Section 2.2 and investigates them from various perspectives. Some work focuses on applications such as a federated system of systems [BGS92] or

logistics procurement activities [NK93], both of which impose centrally-controlled security solutions. Others compose independent systems with certain security properties, yielding a larger system that manifests the same security properties [JM92; McC87; McC88; AL93; Mil94]. Analysis of properties and their composition is accomplished from various viewpoints such as the state-based [BY94], trace-based [McL92], or application-oriented [GG99] perspectives. There is enough variance among these properties and their composition that some have focused on the categorization and classification of these security elements [FG95; FG01; Zak96]. Regardless of the approach, however, two principles are particularly important to maintain when composing policies [GQ94]:

Autonomy: an access allowed by a policy must be allowed by the composition, and

Security: a composition must forbid any access forbidden by a component policy.

Contradictory results are possible if one policy permits access that another disallows. Therefore, during policy composition there must be a mechanism to handle this possibility.

The composability of noninterference policies has been analyzed with consideration of the principles of deducibility security and restrictiveness [McC88]. McCullough determined that deducibility is not a composable property; that is, one cannot make assertions about the deducibility of a system composed of deducibility-secure components [McC90]. Restrictiveness addresses that problem, providing rules that identify when high-level activities have no impact on low-level states, actions, or values. This research focuses on security policy composability with respect to a query-based WSN.

2.3.2 A Policy Language. A number of methods for expressing security policies have been developed [JSS97; CC97; UBJ⁺04; RN02; RZFG99]. [JSS97] identifies *subjects* as those entities to whom authorizations might belong, the things to be done as *actions*, and the items on which the actions are performed as *objects*. In this *Authorization Specification Language* (ASL), constants, variables, predicates, and rules designed around sets of these entities are used to generate authorization specifications.

The types of entities used in ASL are presented in Table 6 along with the variable symbols representing sets of variables associated with the types. Subjects might act as individual entities (*users*) or may act as members of *groups* or in certain *roles*. Groups have certain permissions and are relatively static collections of subjects; subjects cannot choose when to be a part of a group and when to not be (similar to *types* in the object-oriented paradigm). In a relationship similar to subject groups, objects may be operated on because of the individual object's properties or because of the type of object it is. Roles, in contrast, can be activated and deactivated and vary over time. V_r is the variable set of roles while V_R is the variable set for the powerset of roles. Examples of the

Table 6: ASL Types and Variables

Type	Symbol	Description	Associated Variable Set
	A	actions	V_a
	G	groups	V_g
	Obj	objects	V_o
	R	roles	V_r, V_R
	S	subjects	V_s
	T	object types	V_t
	U	users	V_u
sign		+ for authorized, – for denied	n/a
CS		constant symbols	n/a
PS		predicate symbols	$L(v_1, \dots, v_i)$
SA		signed actions	V_{sa}
VS		variable symbols	n/a

uses of the last two variable sets are $r \in V_r$ as in the **active** predicate and $R \in V_R$ as in the **grant** predicate, both discussed below.

Predicates are a more complex ASL type. A predicate has zero or more parameters as indicated by subscripts $a, g, o, r, R, s, s_1, s_2, t$, and u in the table. These parameters may be variables or constants and are included as prescribed by the particular predicate definition. Table 7 presents some possible predicates with their definitions.

Table 7: ASL Predicates

Predicate Expression	Semantics
cando ($s, o, +a$)	Authorization for subject $s \in V_s$ to perform action $a \in V_a$ on object $o \in V_o$ as explicitly specified by the security administrator
dercando ($s, o, +a$)	Derivation expressed the same as cando ; results from some derivation using other predicates and inference
do ($s, o, +a$)	Resolution rule that identifies an authorization the system must recognize; can resolve conflicts between cando and/or dercando
grant ($o, u, R, +a$)	Subject u with active role set $R \in V_R$ may perform action $a \in V_a$ on object $o \in V_o$ (this predicate type enforces the access control policy)
done (o, u, R, a, n)	The n^{th} action (where $n \in \mathbb{N}$) was $a \in V_a$, was performed on object $o \in V_o$ by subject $u \in V_u$ with active roles $R \in V_R$
active (u, r)	Role $r \in V_r$ is active for subject $u \in V_u$
dirin (s, g)	Subject(s) $s \in V_s$ is directly a member of group $g \in V_g$
in (s, g)	Subject(s) $s_2 \in V_s$ is a member of group $g \in V_g$, but perhaps not a direct member
typeof (o, t)	Object $o \in V_o$ is of object type $t \in V_t$
error ()	If error () can be derived there is a problem with the specification or use of authorizations

Predicates can be incorporated into language rules. For example, $\text{cando}(s, o, \langle \text{sign} \rangle a) \leftarrow L_1 \& \dots \& L_n$ identifies that subject s can (or cannot, depending on sign) perform action a on object o when expressions L_1 through L_n hold. The symbology $X \leftarrow Y$, used by [JSS97], is converted to $Y \Rightarrow X$ below, indicating the elements X are true when the elements of Y hold.

To simplify the presentation in this section, wherever they appear, $s \in V_s$, $s' \in V_s$, $o \in V_o$, $o' \in V_o$, and $a \in V_a$. Similarly $s_i \in V_s$, $o_i \in V_o$, and $a_i \in V_a$ where $i \in \mathbb{N}$ and \mathbb{N} is the positive integers.

Language rules are used to create policies that can be composed with other access control policies. Details about policy composition are discussed in Sections 2.3.3 and 2.3.4, but two simple examples of integrity policies are presented below.

Policy A. Policy A is defined in the environment described in Table 8 by a system security administrator to have two integrity levels, Low (L) and High (H). There is one object that can be manipulated by subjects, file F1 on the Low side (of type LFile), and two subjects, SA1 on the Low side (in group L) and SA2 on the High side (in group H). A formal description of an integrity-sustaining policy follows with the predicate representation followed by a plain text description of that predicate.

Table 8: Populated ASL Types for Sample Policy A

Symbol	Value(s)
A	{read, write}
G	{L, H}
Obj	{F1}
R	{}
S	{G, R, U} \equiv {L, H, UA1, UA2}
T	{LFile, HFile}
U	{UA1, UA2}
sign	{s s \in {+, -}}
CS	{LFile, HFile, F1, UA1, UA2, read, write, N}
PS	{cando, dercando, do, grant, done, active, dirin, in, typeof, error}
SA	{(sign) a a \in A}
VS	{ $V_o, V_t, V_g, V_r, V_R, V_a, V_{sa}, V_s$ }

$\text{dirin}(\mathbf{UA1}, \mathbf{L})$ — user UA1 is in group L

$\text{dirin}(\mathbf{UA2}, \mathbf{H})$ — user UA2 is in group H

$\text{typeof}(\mathbf{F1}, \mathbf{LFile})$ — F1 is an LFile, understood to be on the Low side

cando($s \mid s \in \{\mathbf{L} \cup \mathbf{H}\}$, **LFile**, **+write**) — any subject in groups L or H is permitted to write to an LFile

cando($s \mid s \in \{\mathbf{L}\}$, **LFile**, **+read**) — any subject in group L is permitted to read an LFile

cando($s \mid s \in \{\mathbf{H}\}$, **LFile**, **-read**) — any subject in group H is not permitted to read an LFile

In addition to the security administrator's requirements, three rules are used to complete the system's definition. The first two, derivation rules, identify that an authorization is derived when the specified **cando** and **in** predicates both hold; they rely on type membership to deduce authorizations.

DRule 1: $\text{cando}(g, o, a) \wedge \text{in}(s, g) \Rightarrow \text{dercando}(s, o, a)$

DRule 2: $\text{cando}(s, k, a) \wedge \text{typeof}(o, k) \Rightarrow \text{dercando}(s, o, a)$

These rules are applied simultaneously to the security administrator's requirements definition and results in four derived authorizations where each predicate's triple is a system authorization:

dercando(**UA1**, **F1**, **+write**)

dercando(**UA2**, **F1**, **+write**)

dercando(**UA1**, **F1**, **+read**)

dercando(**UA2**, **F1**, **-read**)

These system authorizations confirm what the security administrator had intended; the authorization policy can be expressed this way:

(**UA1**, **F1**, **write**) \rightarrow **authorized**

(**UA2**, **F1**, **write**) \rightarrow **authorized**

(**UA1**, **F1**, **read**) \rightarrow **authorized**

(**UA2**, **F1**, **read**) \rightarrow **denied**

The third rule completes the policy. It is a resolution rule that specifies a lack of positive access to an object implies negative access:

RRule 1: $\neg(\text{cando}(s, o, +a) \vee \text{dercando}(s, o, +a)) \Rightarrow \text{dercando}(s, o, -a)$

This rule states the negative authorization is implied when a positive authorization is not present. This rule doesn't result in further system authorizations in this example, but does establish a fail-safe default condition.

Policy B. Similar to Policy A, Policy B is defined by a security administrator per the domain in Table 9. This policy has three security levels, Top Secret (TS), Secret (S), and Unclassified (U). Three subjects, SB1 (in group U), SB2 (in group S), and SB3 (in group TS), each operate in one of the security levels, U, S, and TS, respectively. There is one file in the system, F2, that is of the type SFile (at security level S). What follow are the SA's formal definition of this system and the results.

Table 9: Populated ASL Types for Sample Policy B

Symbol	Value(s)
A	{read, write}
G	{U, S, TS}
Obj	{F2}
R	{}
S	{G, R, U} \equiv {U, S, TS, UB1, UB2, UB3}
T	{UFile, SFile, TSFile}
U	{UB1, UB2, UB3}
sign	{s s \in {+, -}}
CS	{UFile, SFile, TSFile, F1, UA1, UA2, read, write, \mathbb{N} }
PS	{cando, dercando, do, grant, done, active, dirin, in, typeof, error}
SA	{(sign) a a \in A}
VS	{ $V_o, V_t, V_g, V_r, V_R, V_a, V_{sa}, V_s$ }

dirin(UB1, U) — user UB1 is in group U

dirin(UB2, S) — user UB2 is in group S

dirin(UB3, TS) — user UB3 is in group TS

typeof(F2, SFile) — F2 is an SFile, understood to be in the S domain

cando(s | s \in {U \cup S \cup TS}, UFile, +write) — any subject in groups U, S, or TS is permitted to write to a UFile

cando(s | s \in {U}, UFile, +read) — any subject in groups U or S is permitted to read a UFile

cando(s | s \in {S \cup TS}, SFile, +write) — any subject in groups S or TS is permitted to write to an SFile

cando(s | s \in {U \cup S}, SFile, +read) — any subject in groups U or S is permitted to read an SFile

cando($s \mid s \in \{\mathbf{TS}\}$, **TSFile**, **+write**) — any subject in group TS is permitted to write to a TSFile

cando($s \mid s \in \{\mathbf{U} \cup \mathbf{S} \cup \mathbf{TS}\}$, **TSFile**, **+read**) — any subject in group U, or S, or TS is permitted to read a TSFile

The above rules are positive authorizations, relying on RRule 1 to make the policy complete. An alternative is to explicitly identify the negative authorizations within the policy:

cando($s \mid s \in \{\mathbf{S} \cup \mathbf{TS}\}$, **UFile**, **-read**) — any subject in groups S or TS is not permitted to read a UFile

cando($s \mid s \in \{\mathbf{TS}\}$, **SFile**, **-read**) — any subject in group TS is not permitted to read an SFile

cando($s \mid s \in \{\mathbf{U}\}$, **SFile**, **-write**) — any subject in group U is not permitted to write to an SFile

cando($s \mid s \in \{\mathbf{U} \cup \mathbf{S}\}$, **TSFile**, **-write**) — any subject in groups U or S is not permitted to write to a TSFile

The same three rules as presented with Policy A, DRules 1 and 2 along with RRule 1, apply. Applying these to the policy defined above yields six derived authorizations that are each predicates with system authorization triples as their arguments:

dercando(**UB1**, **F2**, **-write**)

dercando(**UB2**, **F2**, **+write**)

dercando(**UB3**, **F2**, **+write**)

dercando(**UB1**, **F2**, **+read**)

dercando(**UB3**, **F2**, **+read**)

dercando(**UB3**, **F2**, **-read**)

These system authorizations again confirm what the security administrator had intended. Using a notation indicative of a relation between each triple and the set $\{\mathbf{denied}, \mathbf{authorized}\}$ the authorization policy can be expressed this way:

(**UB1**, **F2**, **write**) \rightarrow **denied**

(**UB2**, **F2**, **write**) \rightarrow **authorized**

(**UB3**, **F2**, **write**) \rightarrow **authorized**

(UB1, F2, read) \rightarrow authorized

(UB2, F2, read) \rightarrow authorized

(UB3, F2, read) \rightarrow denied

2.3.3 A Policy Composition Framework. A policy composition framework should have the following characteristics if that framework is to effectively support the composition of access control policies [BdVS02]:

Heterogeneous policy support is the ability of the framework to combine not only a wide variety of policies but also a wide variety of languages.

Support of incomplete policies accounts for policies in which part of the specification is unknown.

Provision for unknown policies uses placeholders where a policy is known to exist, but the policy itself is unknown until run-time.

Controlled interference provides a means for merging policies when a direct mapping of rules is impossible.

Expressiveness mandates that a language used within the framework express a wide range of policy specifications without language extensions.

Support of different abstraction levels is required for efficient and effective policy design, analysis, and refinement.

Formal semantics implies a declarative language that ensures analysis with proofs and design verification are possible.

The terms used in a composition framework consist of a fixed set of subjects S , objects O , and actions A . Terms are of the form (s, o, a) where s , o , and a correspond to an element in S , O , and A , respectively, with the meaning that s is authorized to perform action a on object o . A subject s can, by its properties, be a role, group, or user. Similarly object o can function as an object or type. The definition of a *policy* is “a set of ground authorization terms” where a *ground term* is a variable-free triple requiring no further interpretation to determine its semantic meaning. Such terms are referred to as *system authorizations* hereafter. For a dynamic system this means the policy is also dynamic, responding differently depending on the roles that are active at the moment of any given request.

The structure of authorization constraint languages \mathcal{L}_{acon} may vary, but can be represented by predicates using no terms other than those found in the sets S , O , and A [BdVS02]. The relation

$\text{satisfy} \subseteq \mathbf{S} \times \mathbf{O} \times \mathbf{A} \times \mathcal{L}_{acon}$ identifies whether a given triple (s, o, a) satisfies the given policy, where s , o , and a are, respectively, elements of \mathbf{S} , \mathbf{O} , and \mathbf{A} .

Similarly, a rule language \mathcal{L}_{rule} may vary but the closure relation $P(\mathcal{L}_{rule}) \times P(\mathbf{S} \times \mathbf{O} \times \mathbf{A}) \rightarrow P(\mathbf{S} \times \mathbf{O} \times \mathbf{A})$ are the permissions that can be derived by each rule per authorizations found in a given policy.

Using ASL as the basis for an authorization constraint language \mathcal{L}_{acon} , \mathbf{S} is the set of subjects \mathbf{S} , \mathbf{O} is the union of the sets of objects \mathbf{O} and types \mathbf{T} , and \mathbf{A} is the set of actions \mathbf{A} . \mathcal{L}_{acon} is comprised of the described ASL predicates (e.g., `cando` and `dirin`). Similarly, ASL rules are an example of a rule language \mathcal{L}_{rule} .

To compose Policies A and B from Section 2.3.2, the various sets discussed here are populated. $\mathbf{S}_A = \{\text{L, H, UA1, UA2}\}$, $\mathbf{O}_A = \{\text{LFile, HFile, F1}\}$, $\mathbf{A}_A = \{\text{read, write}\}$, $\mathbf{S}_B = \{\text{U, S, TS, UB1, UB2, UB3}\}$, $\mathbf{O}_B = \{\text{UFile, SFile, TSFile, F2}\}$, and $\mathbf{A}_B = \{\text{read, write}\}$. \mathcal{L}_{acon} is translated as the elements of Table 7 dictate with pertinent expressions within the language given in the discussions of the policy examples. Using the predicates available in \mathcal{L}_{acon} , the constants and variables for Policies A and B, and \mathcal{L}_{rule} , and the constructs from first-order logic, each rule in \mathcal{L}_{rule} is of the form $\langle \text{predicate expression} \rangle (\langle \text{logic symbol} \rangle \langle \text{predicate expression} \rangle)^* \Rightarrow \langle \text{predicate expression} \rangle$ where logic symbol is from the set $\{\wedge, \neg\wedge, \vee, \neg\vee\}$ as exemplified in the rule $\text{in}(s, H) \wedge (\text{cando}(s, F1, \text{read}) \vee \text{dercando}(s, F1, \text{read})) \Rightarrow \text{dercando}(s, F2, \text{read})$.

Having the framework for policy composition, the discussion turns to the algebra used for performing the composition process.

2.3.4 An Policy Composition Algebra. Any algebra used for access policy composition must express the given policies and have the capability to compose them in various ways [BdVS02]. With the assumption that the policies are represented as in Section 2.3.3, consider the following policy manipulations where P_1 , P_2 , and P_3 are policies and P is a partially specified policy.

Addition is left-associative, is the union of two policies denoted $P_1 + P_2$, and has precedence 2.

Conjunction is left-associative (precedence 2), is the intersection of two policies, and is denoted $P_1 \& P_2$.

Subtraction is left-associative (precedence 3) and eliminates restrictions found in one policy from another. It is denoted $P_1 - P_2$.

Closure is left-associative with precedence 3. Closure applies a set of inference rules to a policy to close that policy. The term *closure* refers to the complete enumeration of permitted actions

according to the facts and rules presented in the language and is denoted $P_1 * R$, where R are rules expressed as Horn clauses.

Scoping restrictions are non-associative with precedence 3. These limit the application of a policy to a specific set of subjects, objects, and actions, and are denoted $P_1 \hat{c}$ where c is the set of subjects, objects, and actions.

Overriding is non-associative (precedence 1). It requires three policies combined via a subtraction, conjunction, and addition (given policies P_1 , P_2 , and P_3 : subtract P_3 from P_1 and then add the conjunction of P_2 and P_3). It is denoted $o(P_1, P_2, P_3)$ and is shorthand for $(P_1 - P_3) + (P_2 \& P_3)$.

Template is non-associative with precedence 0. Templates provide for incomplete policy specifications, denoted as τP where τ is the symbol indicating P is a template. Templates are beyond the scope of this research.

Modification of policy environment e indicated by $e[P_1/P_2](P_3)$ means that the environment is defined by P_1 if $P_3 = P_2$ but is defined by P_3 otherwise. Modification is not addressed in this research.

The higher precedence operator in an equation is executed first and the lowest, last. In this scheme the highest precedence is 0 and the lowest is 3. Associativity describes the order of operations in a binary operation in the absence of explicit parentheses. In an expression containing more than one non-associative operator parentheses are required to identify which operator is executed first (e.g., $o(o(P_1, P_2, P_3), P_4, P_5)$). Left-associative operators are understood to be parenthesized from left to right when there are no explicit parentheses (e.g., $P_1 + P_2 + P_3 \equiv (P_1 + P_2) + P_3$).

Expressing policies and providing a means for composing them is necessary, but ultimately policy expressions must be converted into logic for evaluation. Bonatti, et al., develops a method for translating policies into logic programs when the rule language consists of triples (cf. Section 2.3.3). Using this method, an environment, e , maps policy identifiers (labels used to reference specified policies) to sets of system authorizations. For the policy identifier X in environment e , the semantics are denoted $[[X]]_e \stackrel{def}{=} e(X)$ where evaluating X in e identifies the system authorizations, $e(X)$.

Canonical labeling is applied to each policy expression to create a *labeled policy expression* used during translation; this identifies the operator within an expression so it can be evaluated next in the recursively defined translation. Translation of policy E requires the canonical labeling of E , identified as E^ℓ , with the translation of E in environment e into a logic program expressed as $pe2lp(E^\ell, e)$, as is shown in the first line of Table 10. The function $pe2lp$ operates on the input parameters to translate from composition symbology (the policy composition expression) into a logic program (using elements of \mathcal{L}_{acon} and \mathcal{L}_{rule}).

In this table, P is a policy identifier while F , G , and M are policy expressions. R represents a set of inference or derivation rules (e.g., $R = \{(s, o, a) \Rightarrow (s', o, a)\}$) for the policy related through the $*$ operator. L is an authorization term of the form (s, o, a) or a basic predicate p ; these constructs' translations are provided in the last three lines of Table 10, adapted from [BdVS02]. $mainp_F$ denotes the predicate \mathbf{auth}_ℓ , where ℓ is the main label of F . The predicates \mathbf{auth}_i and \mathbf{auth}_P specify the system authorizations identified during translation. Where applicable, the modification corresponding to the symbolic representation of an expression is included in the table.

Table 10: Translation $pe2lp$

Expression (Modification)	Logic Program Translation
E	$pe2lp(E^\ell, e)$
P	$\{\mathbf{auth}_P(s, o, a) \mid (s, o, a) \in e(P)\}$ if $e(P)$ is defined, \emptyset otherwise.
$F +_i G$ (Addition)	$\{mainp_F(x, y, z) \Rightarrow \mathbf{auth}_i(x, y, z), mainp_G(x, y, z) \Rightarrow \mathbf{auth}_i(x, y, z)\}$ $\cup pe2lp(F, e) \cup pe2lp(G, e)$
$F \&_i G$ (Conjunction)	$\{mainp_F(x, y, z) \wedge mainp_G(x, y, z) \Rightarrow \mathbf{auth}_i(x, y, z)\}$ $\cup pe2lp(F, e) \cup pe2lp(G, e)$
$F -_i G$ (Subtraction)	$\{mainp_F(x, y, z) \wedge \neg mainp_G(x, y, z) \Rightarrow \mathbf{auth}_i(x, y, z)\}$ $\cup pe2lp(F, e) \cup pe2lp(G, e)$
$F \hat{ }_i c$ (Scoping)	$\{mainp_F(x, y, z) \wedge c \Rightarrow \mathbf{auth}_i(x, y, z)\} \cup pe2lp(F, e)$
$o_i(F, G, M)$ (Overriding)	$\{mainp_F(x, y, z) \wedge \neg mainp_M(x, y, z) \Rightarrow \mathbf{auth}_i(x, y, z),$ $mainp_G(x, y, z) \wedge mainp_M(x, y, z) \Rightarrow \mathbf{auth}_i(x, y, z)\}$ $\cup pe2lp(F, e) \cup pe2lp(G, e) \cup pe2lp(M, e)$
$F *_i R$ (Closure)	$\{\mathbf{trans}(L_1, i) \wedge \dots \wedge \mathbf{trans}(L_n, i) \Rightarrow \mathbf{auth}_i(s, o, a)$ $\mid (L_1 \wedge \dots \wedge L_n) \in R \Rightarrow (s, o, a)\}$ $\cup \{mainp_F(x, y, z) \Rightarrow \mathbf{auth}_i(x, y, z)\} \cup pe2lp(F, e)$
L	$\mathbf{trans}(L, i)$
(s', o', a')	$\mathbf{auth}_i(s', o', a')$
$p(x_1, \dots, x_n)$	$p(x_1, \dots, x_n)$

An example helps clarify the terminology. Policy $E = P + Q + o(Q, R, S + T)$ is first labeled with sequential numbering of E 's operators as in $P +_0 Q +_1 o_2(Q, R, S +_3 T)$. Next the *main label* of a policy expression X , $mainp_X$, identifies the expression's root, or final, operator. Given associativity and precedence rules for E , the main label is $+_1$; this is made clear by analyzing the fully parenthesized expression $(P +_0 Q) +_1 o_2(Q, R, S +_3 T)$. The operator identified by the main label is the first used to translate policy expressions into logic programs as presented in Table 10. For this example $mainp_E$ is $+_1$. From the translation table $F +_1 G$ applies where $F \equiv P +_0 Q$ and $G \equiv o_2(Q, R, S +_3 T)$.

Iterative application of the translation rules develops the set of system authorizations for the given policy composition. This logic program representation evaluates the permissibility of activities within the context of the desired policy composition. A request for the execution of a given action, represented by a triple (s, o, a) , is checked against the entries in the logic program equivalent of the **satisfy** relation; the action is permitted only when $(s, o, a) \in pe2lp(E^\ell, e)$.

Consider the following example of a composition algebra and translation of the resultant policy into a logic program. Suppose Policy A (PA) and Policy B (PB) from Section 2.3.3 are implemented on adjacent systems that must now work together. First the system authorizations are delineated:

For PA

$$\begin{array}{l|l} (SA1, F1, +write) & (SA1, F1, +read) \\ (SA2, F1, +write) & (SA2, F1, -read) \end{array}$$

leads to a relation **satisfy** $\subseteq S \times O \times A \times \mathcal{L}_{acon} = \{(SA1, F1, write), (SA2, F1, write), (SA1, F1, read)\}$

For PB

$$\begin{array}{l|l} (SB1, F2, -write) & (SB1, F2, +read) \\ (SB2, F2, +write) & (SB3, F2, +read) \\ (SB3, F2, +write) & (SB3, F2, -read) \end{array}$$

similarly results in relation **satisfy** $\subseteq S \times O \times A \times \mathcal{L}_{acon} = \{(SB2, F2, write), (SB3, F2, write), (SB1, F2, read), (SB3, F2, read)\}$

The security administrator determines that L in Policy A is equivalent to U in Policy B, H is similarly equivalent to S, and TS remains higher than all others. The rule to implement this is

$$R = \{\mathbf{in}(V_u, L) \Rightarrow \mathbf{in}(V_u, U), \mathbf{in}(V_u, H) \Rightarrow \mathbf{in}(V_u, S)\}.$$

The intended closure of the composed properties incorporates the original system authorizations and adds terms as determined by the application of the new rules. The algebraic expression of this composition is as $PC = (PA + PB) * R$, where PC is the policy composition of PA and PB.

Applying rule $\mathbf{in}(V_u, L) \Rightarrow \mathbf{in}(V_u, U)$ leads to

$$\text{newDercando1} = \{\mathbf{dercando}(SA1, F2, -write), \mathbf{dercando}(SA1, F2, +read)\}$$

while rule $\mathbf{in}(V_u, H) \Rightarrow \mathbf{in}(V_u, S)$ leads to

$$\text{newDercando2} = \{\mathbf{dercando}(SA2, F2, +write), \mathbf{dercando}(SA2, F2, +read)\}.$$

$(PA + PB) * R$ is rewritten $(PA +_0 PB) *_1 R$ in preparation for translation $pe2pl((PA +_0 PB) *_1 R) = (pe2pl(PA, e) \cup pe2pl(PB, e)) \cup pe2pl(\text{newDercando1}) \cup pe2pl(\text{newDercando2})$. All of these manipulations yield the following policy with the understanding that any triples not specified are not authorized:

authPA(SA1, F1, write) → authorized ,
authPA(SA2, F1, write) → authorized ,
authPA(SA1, F1, read) → authorized ,
authPB(SB2, F2, write) → authorized ,
authPB(SB3, F2, write) → authorized ,
authPB(SB1, F2, read) → authorized ,
authPB(SB2, F2, read) → authorized ,
auth1(SA1, F2, read) → authorized ,
auth1(SA2, F2, write) → authorized , and
auth1(SA2, F2, read) → authorized .

2.4 *Related Research*

The above sections describe technologies directly related to this research. This section discusses related research efforts: WSN security, policy composition, and policy languages.

Security for WSN's has garnered a great deal of attention. Some articles focus on identifying areas of WSN security that still require a great deal of research [Sta04; PSW04]. Much of the current research is focused on encryption protocols and other data manipulation methods to promote confidentiality, integrity, and provide effective authentication [ZG04; RSZ04; KSP06]. Encryption key management appears prominently in related research efforts, a few of which are identified below.

One effort among many seeking key predistribution schemes to augment security in WSNs is [DDH⁺05], shown to have a high probability of non-compromised communications in a WSN when the number of compromised nodes is less than a certain percentage (33%) and when the number of hops required for communication is limited (≤ 3 hops).

A security architecture called SPINS [PSW⁺01] establishes a secret key between WSN nodes using a gateway as a trusted third party. Communications subsequent to key establishment do not require third party involvement. The key helps ensure the confidentiality of communications within the WSN.

The Reputation-Based Framework for Sensor Networks [GS04] notes cryptography's limitations with respect to certain internal attacks or failures and addresses these by observing node be-

havior to determine trustworthiness and counter various types of possibly adverse activities within the network.

Some research related to WSN security is not WSN-specific, but focuses on the concealing of policy information within environments such as a WSN using one-way hash functions [KHJ02]. One-way hash functions protect policies based only on a current event rather than on a sequence of events. Other efforts assume an adversary will infiltrate the WSN and attempt to identify and eliminate such a threat [GS04; AIL05]. No related work appears to implement a dynamic security policy management within a WSN.

Policy composition is approached from various perspectives. Some research focuses on or manipulates specific security properties [McC88; Zak96] with others using the term *composition* to address the joining of two systems exhibiting the same security property [McC87; JM92].

Similarly, [AL93] analyzes components, their policies, and the effects of linking the components, and thus their policies, together. These policy composition research efforts analyze event traces or state models to determine the accuracy or effectiveness of the composition. In contrast, the research presented herein composes stateless policies and considers previous events and the current status of the system as irrelevant to whether or not a particular request is granted.

Languages expressing policy and policy compositions vary widely and are sometimes coupled with particular applications. *Ponder* [DDLS01] is an object-oriented-style methodology for specifying policies. It specifies management functions for distributed networks and thus avoids formal-language modeling while providing a widely extensible and flexible framework for policy specification.

Law-Governed Interaction [AMN02] addresses enterprise system management, and requires layered security policies and subsequent interaction of those policies. Policy compositions in this system are possible because each policy is a descendent of a common superior policy. Subordinate policies and laws thus structurally conform to the same set of rules established at the higher level.

Binder [DeT02] is a logic-based security language to communicate security requirements via Prolog-like logic statements. This approach uses an open system to express security requirements for distributed systems, not concealing the security statements.

Other languages exist, but the one used herein [JSS97] provides formalisms for analysis and lends itself to analysis of system authorizations. These characteristics are helpful in accomplishing the research objectives within the context of a WSN.

III. WSN Authorization Specification Language

Any policy-enforcement system must have a means for expressing the policies—a policy specification language. This language should be flexible to express a variety of policy types as well as a number of implementations. It must also have sufficient structure to permit a formal examination of policy properties and identify adherence to stated requirements. An additional requirement is imposed by a WSN; evaluation of policies specified in the language must produce an expression of the security policy that is compatible with the limited resources of a WSN node. The WSN Authorization Specification Language (WASL), described below, provides an effective representation for a wide variety of policies including the policy types mentioned herein and provides the means for composition, as well.

A WASL policy, with the formal basis of the language drawn from ASL [JSS97], is defined using three kinds of statements: identifiers (constant or variable terms), relational statements, and rules. The type-hierarchy of statements is shown in Figure 4; the † indicates an abstract supertype—a supertype of which there are no specific instances.

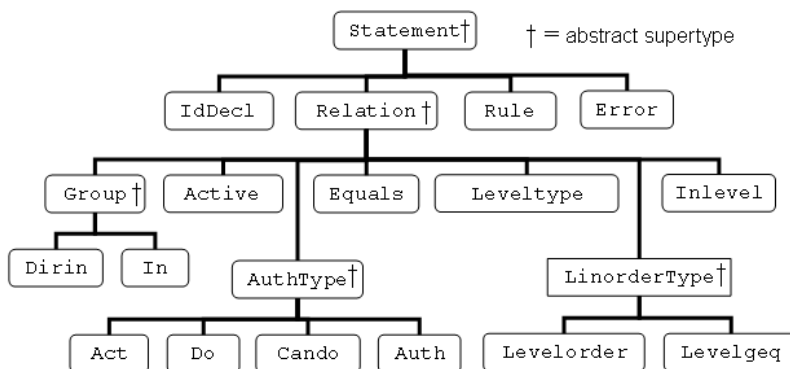


Figure 4: Statements in WASL

A policy is expressed in WASL as a set of **Statements** contained within a **Block**. Alpha-numeric constants and variables identify the system’s entities and must be declared as **terms** of **IdDecl**; **Relations** identify relationships among terms and are expressed as $r(t_1, \dots, t_n)$ where r is the relation symbol and t_i are the constant or variable terms appropriate for r . The primary **Relations** are the statements **Dirin**, **In**, **Active**, **Equals**, **Leveltype**, **Inlevel**, **Levelorder**, **Levelgeq**, and **Done**. A **Rule** asserts that when some condition is met, the **Relational** consequence in the the **Rule** is true; variable terms are found only in **Rules**. **Error** is a special **Statement** that is a consequence of some **Rules** or when specification errors are identified; the presence of this statement in the policy indicates a contradiction in the assertions of the policy.

Authorizations are expressed through the **Relation** statements called **AuthTypes**: **Act**, **Do**, **Cando**, and **Auth**. Discretionary authorizations (authorizations based on the system administrator’s deter-

mination) are expressed using `Act` (used solely for role-based policies) and `Cando` while `Do` enumerates mandatory (rule-based) authorizations. `Auth` is used to express the authorizations for a system. All of the above statement types establish the policy used to determine a system's authorizations; these terms are expressions of the authorizations of each subject to the various objects using the `Auth` statement type.

The most significant differences between WASL and ASL [JSS97] include the following, with some language elements added to focus on security levels and their hierarchy:

- Relations `do` and `cando` represent mandatory and discretionary accesses respectively (e.g., `do(Subj1,Obj2,Read)`).
- Relation `auth` expresses system authorizations (authorization decisions, e.g., `auth(Subj1,Obj2,Write)`).
- Type `level` identifies the system's security levels (e.g., `const level Unclass`).
- Type `leveltype` identifies the category of security levels as associated with the `leveltype` relation (e.g., `const leveltype BLP; leveltype(Unclass,BLP)`).
- Relation `dirin(e,g|k)` is used to provide a nested hierarchy of capabilities and identities for both subjects and objects.
- Relation `inlevel(e,l)` associates entity `e` (a `subject`, `group`, `object`, or `kind`) with `l` (a `level`).
- Relation `levelorder(l1,l2)` orders security levels where `l1` is "higher than" `l2`.
- Relation `levelgeq(l1,l2)` identifies `level l1` as "greater than or equal to" `level l2`.
- Relations `dercando`, `done`, `grant`, and `kindof` are not used.

Additionally, whereas ASL expresses facts implicitly within rules, WASL generates an exhaustive, explicit expression of authorizations. This listing requires the assertion of facts via relational statements.

Section 3.1 presents the environment WASL is assumed to be implemented in and continues with a comprehensive presentation of the language in Section 3.2.

3.1 Network Under Study

This section examines the limitations of WSNs that lead to particular design decisions. For example, due to bandwidth, memory, and processing power limitations nodes do not perform compilation or composition; these functions are performed at the gateway. WSN characteristics are presented in Section 3.1.1 while policy-related responsibilities are discussed in Section 3.1.2.

3.1.1 System Characteristics. The WSN under consideration is a query-based system with dozens to thousands of resource-constrained nodes and a more powerful gateway computer. Each of these characteristics impacts implementation decisions discussed in Section 3.1.2.

In a query-based system information is transferred primarily as a result of requests for data. This can also be thought of as a *pull*-oriented rather than *push*-oriented system; the node does not automatically send its data to nearby nodes but instead responds to queries for data. Queries may be received by any network node and, if the query arrives at a node other than the intended destination, it is routed appropriately.

Queries may originate at nodes external to the WSN. No assumptions are made about these nodes other than that they are capable of communicating with the nodes in the WSN, including authentication with a verification of security properties. The external nodes may be participating in a network that functions under a security policy distinct from that of the WSN. The provision for handling queries when the querying subject is external to the WSN is addressed by composition (Chapter V).

Each node in the WSN may respond to a query and participate in any query-response exchange as a message router in the network. Given that messages may be classified or protected at different security levels, it is assumed that each node has a trusted kernel, secure to the highest level of security in the WSN. Thus, regardless of the security level at which a node functions, the trusted kernel can process messages at higher security levels.

The gateway is a laptop-class node that functions as the administrative hub of the WSN. Policy or other software updates are distributed from this node and it generally functions as the interface between the WSN and external networked devices. Because the WSN may be highly distributed, it is not assumed that it can communicate directly with each of the nodes in the network. Nor is it assumed that a response from a node must pass through the gateway before being received by an external node.

Data communicated to, from, or within the network is encrypted at an appropriate level and all subjects adhere to an authentication scheme. Each query is thus secure with respect to confidentiality and integrity while in transit. Additionally, because the querying subject is properly identified, policy information can be effectively used to determine appropriate responses. The policy itself is expected to change over time with subjects or groups being added or their authorizations modified. But these changes are assumed to occur infrequently.

3.1.2 Distribution of Responsibilities. The primary policy-related tasks include the following:

1. Controlling the WSN security policy specification (by the security administrator),
2. Compiling the WSN's policy,
3. Collecting external networks' possibly partial policy specifications,
4. Composing the WSN's policy with other policies,
5. Compressing the resulting system authorizations for distribution and storage,
6. Distribution of system authorizations, and
7. Storage of system authorizations.
8. Choosing the proper response when queried.

The minimal capabilities of nodes necessarily requires that many of these tasks be performed at the gateway. Additionally, while the administrator has access to the gateway, system authorizations are distributed to the nodes via wireless communications. The gateway, therefore, is the sole node in the system that performs tasks 1 through 5. These tasks include the original WASL expression of the policy, the compilation of the policy, and the management of the tasks related to establishing a secure basis for interactions with external networks' nodes.

Task 6 begins at the gateway, but is also shared by every node in the WSN. It is expected that many nodes will receive data and system authorizations to be retransmitted to other nearby nodes. Each node must store at least those system authorizations that are pertinent to its own operations or data accesses; this is expressed in task 7.

Tasks 7 and 8 could be implemented in two ways. One is to have a node store every system authorization. The other is to store only those system authorizations related to objects maintained at the node. The following discussion addresses the options. In either case, all the system authorizations must be distributed throughout the WSN so the nodes have all the system authorizations they need.

For the first alternative an obvious impact is the use of a node's memory. Storing all authorizations requires significantly more memory but has the benefit of mitigating a class of denial of service attacks; when a node has access to every system authorization it determines whether to respond to or forward a query. The assumption is that nodes ignore queries by entities not specifically authorized by the security policy, thus reducing the number of query transmissions within the WSN at the cost of increased memory usage and slightly more computation by a node.

The second alternative reduces the memory required at each node, potentially significantly, depending on the number of authorizations and the scheme used for system authorization representation. This decision, however, means each node will forward any queries intended for another node which may result in a much larger energy expenditure due to the forwarding of unauthorized queries.

Every node performs task 8—every query is evaluated with respect to the system authorizations prior to the generation of a system response. For authorized queries, the queried node performs the appropriate computations and/or transmissions in response to the query as appropriate. Generally it is expected that answers to queries are routed to the gateway for transmission to inter-network nodes as appropriate because nodes have a very limited effective transmission range. It is possible, however for an inter-network node to receive the answer directly from another node. Queries are assumed to be routed to their intended destination where they are compared to system authorizations.

3.2 *WASL: The Language*

The above description of the targeted network establishes a frame of reference for policy language design. Section 3.2.1 presents the grammar and syntax and is followed by respective descriptions of the semantics of terms, relations, and rules in Sections 3.2.2, 3.2.3, and 3.2.4. Section 3.2.5 discusses the process of compilation.

3.2.1 Grammar. A detailed specification of WASL grammar with accepted inputs is presented in Table 11. This grammar specifies all the inputs that can be parsed by the WASL compiler. **Courier** font in the table identifies terminal symbols while italics identifies nonterminals. Alternatives are captured within square brackets with options separated by the vertical bar. A question mark (?) following the brackets of an alternative identifies an optional entry, a superscript asterisk (*) identifies any number of (zero or more) sequential appearances of the alternative, and a superscript plus sign (+) identifies there are one or more appearances. The terminal **str** is specified such that any number of any characters may be used, with the exceptions of newline symbols and the period (.). **str** terminates with a period.

A policy expressed in WASL begins with **begin** and ends with **end;**. Additionally, every statement is terminated with the semicolon (;). Just because a program can be parsed does not guarantee the input is acceptable, but grammatical requirements guarantee a structure that can be analyzed for proper semantics as described in other sections.

A convention adopted in this paper but not mandated by the grammar is to use terms beginning with a capital letter for constants. Variable terms' first characters are lowercase.

3.2.2 Declarations of Terms. Every WASL term is of an **IdType** of **role**, **level**, **leveltype**, **actor**, **group**, **subject**, **target**, **object**, **kind**, or **action**. As the hierarchy shown in Figure 5 illustrates, the **subject** and **group** types specialize the **actor** and, similarly, the **kind** and **object** specialize **target**. System entities include both **actors** and **targets**. The † indicates **entity** terms are supertypes that cannot be instantiated while the ‡ identifies **actor** and **target** terms supertypes that can be

Table 11: WASL Grammar

Element \rightarrow Construction
<i>Policy</i> \rightarrow begin [<i>Stm</i> ;]* end ;
<i>Stm</i> \rightarrow [<i>IdDecl</i> <i>RelStm</i> <i>ErrorStm</i> <i>RuleStm</i>]
<i>IdDecl</i> \rightarrow [const var] [action actor group kind level leveltype object role subject target] id
<i>ErrorStm</i> \rightarrow error (str)
<i>RelStm</i> \rightarrow [act (id , id , [+ -]? id [, id]+) active (id , id) auth (id , id , [+ -]? id [, id]*) cando (id , id , [+ -]? id) dirin (id , id) do (id , id , [+ -]? id) equals (id , id) in (id , id) inlevel (id , id) levelgeq (id , id) levelorder (id , id) leveltype (id , id)]
<i>RuleStm</i> \rightarrow <i>Exp</i> \Rightarrow <i>RelStm</i> <i>ErrorStm</i>
<i>Exp</i> \rightarrow [true <i>Exp</i> [&] <i>Exp</i> [-]? (<i>Exp</i>) [+ -]? <i>RelStm</i>]
id \rightarrow [a - z A - Z] [a - z A - Z 0 - 9]*
str \rightarrow (~ [\n \r .]) * [.]

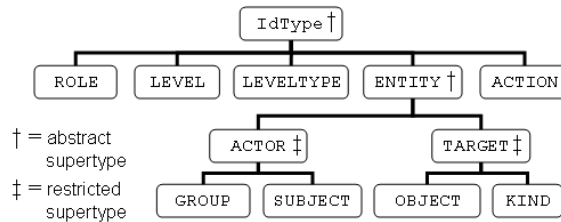


Figure 5: Types of Terms (Identifiers) in WASL

instantiated but are restricted to variable terms (e.g., a `const target` declaration is an error). The special subtype of `action` (`SIGNEDACTION`) is not shown in the figure but attaches a unary operator of `+` or `-` to the action within an `AuthType` statement to signify the given relationship as a positive (permitted) or negative (disallowed) authorization.

The following restrictions provide unambiguous semantics for terms.

- Terms may be declared as either `constants` or `variables`.
- Term labels begin with an alpha character but contain any number of alpha-numeric characters.
- No label in a policy may be defined more than once.
- An `actor` term is always a variable, appearing only in relation expressions (see Section 3.2.4). `Subject` and `group` terms are the constants of this type.
- A declared `role` must be a constant.
- Variable terms may appear only within rules.

3.2.3 Relations. Twelve relations further develop the policy by giving context to the terms and their types. Relations' semantics are defined according to the in-order parameter requirements identified in Table 12. In this table the symbol `*` indicates the associated set may be empty while the `+` identifies a necessarily non-empty set. The discussion below uses the following labels as representatives of the given types: actor `c`, subject `s`, group `g`, target `t`, object `o`, kind `k`, level `l`, leveltype `lt`, action `a`, and signed action `±a` (representing the expression `+a` or `-a`). A term that can be an `actor` or `target` (or any of their subtypes) is identified as `e` while the terms (`id`) that can be of any type are `x1` and `x2`.

- `act(c,t,±a,{r}+)` specifies discretionary access controls with the additional requirement that each role in `{r}` be active.
- `active(s,r)` defines role `r` as active for `s`.
- `auth(c,t,±a,{r}*)` expresses system-authorized access—the system authorizations.
- `cando(c,t,±a)` is used to express discretionary access controls; `c` (or each member of `c`) is allowed (forbidden) to perform `a` on `t`.
- `dirin(e,g|k)` defines `e` as an immediate member of `g` or `k`.
- `do(c,t,±a)` is used to express mandatory access controls; `c` (or each member of `c`) is permitted to perform `±a` on `t` based on a set of rules.

Table 12: WASL Relation Term Requirements

Relation	(Ordered Term Types)
act	(actor, target, signedaction, {role} ⁺)
active	(subject, role)
auth	(actor, target, signedaction, {role} [*])
cando	(actor, target, signedaction)
dirin	(entity, group or kind)
do	(actor, target, signedaction)
equals	(id, id)
in	(entity, group or kind)
inlevel	(entity, level)
levelgeq	(level, level)
levelorder	(level, level)
leveltype	(level, leveltype)

- `equals(x1,x2)` appears only in the conditional expression of rules and identifies that `x1` is the same term as `x2`.
- `in(e,g|k)` defines `e` as a member of `g` or `k`, although not necessarily an immediate member.
- `inlevel(e,l)` defines `e` as being in security level `l`.
- `levelgeq(l1,l2)` appears only in the conditional expression of rules and identifies that `l1` is at the same or a higher security level than `l2`.
- `levelorder(l1,l2)` defines `l2` as the next lower security level than `l1`.
- `leveltype(l1,lt)` defines `l2` as the `lt` type of security level.

Level ordering is different than entity grouping since security levels have a strictly controlled hierarchy. An actor or target may be identified as indirectly belonging to a group or kind (using `in`) while applying the `levelorder` statement indicates the hierarchy of levels. This, along with a check for loops in the level ordering during compilation, ensures a partial ordering of the security levels. Evaluation of the relation `levelgeq`, a relation that appears only in expressions (cf. Section 3.2.4), checks `levelorder` entries to determine whether they are true or false.

The `error` statement is similar to a relation statement, but is unique because it relates no terms. It does, however, have a text field that provides feedback regarding a problem in the policy (e.g., `error(DoStm conflict for do(Subject3,Obj2,-Write)).`).

3.2.4 Rules. The rule construct is fundamentally intended to capture a policy’s mandatory authorizations. For example, it is used to express the relationships between subjects, objects, levels, and actions necessary to sustain the prescribed definition of security. Rules may also be used by the

security administrator, however, to quickly define relationships that authorize or prohibit certain actions.

Rules consist of a relation expression *re* (the condition) and a relation or error statement *rs* (the consequent). Any *re* is a boolean expression of zero or more relations linked together and operated upon by logical operators; unary operators are + (PLUS) and - (MINUS) while binary operators are & (AND) and | (OR). An empty *re* implies *true*. Relations *r* that appear in *re* may have variables and constants as terms or both.

A relation expression $+r$, where *r* is a constant-only relation in *re*, evaluates to *true* if and only if it appears in the policy; it evaluates to *false* otherwise. $-r$ yields the opposite result. Any *r* in *re* with variable terms cannot be evaluated *true* due to the variables, but it can be evaluated *false* if the existing constants do not match the patterns present in the statements of the policy. This handling of relations is the basis for evaluating binary expressions, with & and | providing the logical conjunction and disjunction, respectively.

Resolving rules is accomplished by substituting constant terms from the policy for the rules' variables, ensuring the typing of terms is maintained. The variable-free statement *rs* of a rule is added to the policy when *re* of the same rule evaluates to true.

Several requirements are associated with rules:

- A rule with an **auth** statement in the condition must have an **auth** statement as the consequent. A rule structured this way is used only during composition due to the semantics of the various relation statements.
- The consequent of a rule must be either an authorization type (**do**, **cando**, **act**, or **auth**) or an **error**. All other relations are explicitly specified or inferred during compilation.

3.2.5 Compilation. The WASL compiler performs two functions: compilation of the expression of the policy in WASL (a policy *program*) and policy composition. Compilation leads to the complete delineation of system authorizations with constant-only terms. Composition is addressed in Chapter V.

Compilation of a WASL program generates a set of system authorizations or **auth** statements. Compilation includes lexical analysis, parsing, semantic analysis, and code generation. The textual input is confined to the grammar specified in Table 11 and the output, consistent with the same grammar, includes the original input, statements deduced from the input, and the generated set of authorizations. The sequential steps of compilation follow.

1. Analyze the input for proper grammatical structure and store in an intermediate representation useful for program analysis (lexical analysis and parsing).
2. Bind each term to its declaration to ensure, for example, `Node32` always refers to `const subject Node32` (semantic analysis, part 1).
3. Resolve types; for example, parsing identifies the subject in a `do` statement as an `actor` whereas it may actually be a `subject` or `group` according to its binding. (semantic analysis, part 2).
4. Check structural requirements; for example, a rule must have as its consequent either an `error` or an `authtype` statement, security levels must be arranged in a partial order, and entities must not be in multiple security levels within a given partial order (semantic analysis, part 3).
5. Analyze relations to determine any possible derived `in`, `inlevel`, or authorization relations (semantic analysis, part 4).
6. Identify rules that do not generate system authorizations and iterate through the set of variables they contain.
 - (a) Identify and iterate through constants matching the variable's type.
 - i. Iterate through rules requiring a substitution, replace the variable with the constant to create a new rule.
 - ii. Simplify and evaluate the condition of the created rules.
 - iii. Add to the program consequents corresponding to conditions that are `true`.
 - iv. Store the rules to replace the next variable if further constant-for-variable substitutions are required.
 - (b) Repeat with the next variable in Step 6 using the stored rules from Step 6(a)iv.
7. If any relations have been added to the program, repeat Step 6.
8. Identify rules that generate system authorizations and perform the iterations of Step 6 using these rules.

Policies may express different types of authorizations using a variety of terms: mandatory accesses, discretionary accesses, access control lists, and rule-based accesses are just a few. WASL relations model these using `act`, `do`, and `cando` relations. The generation of system authorizations relies on the proper expression of security rules to specify how the interaction of these constructs results in system authorizations.

Compilation thus results in the desired set of system authorizations identifying the authorizations of the system. This complete expression of system authorizations is sufficient for a system

operating independently of all other systems. However, as soon as interactions with external networks is necessary, policy composition is required.

3.3 Summary

WASL provides a structure for specifying arbitrary security policies. The target network for these policies shapes the language requirements and identifies the need for a complete delineation of system authorizations.

The result is a language with a precise grammar, defined semantics for terms and relations, and the provision for rules that are used to make assertions about groups of entities and their security levels as well as related authorizations. Compilation steps expand a specified policy to include the desired complete set of system authorizations consistent with that specification.

IV. Specifying Policies in WASL

Policy specification using WASL is straightforward and several examples are presented in this chapter. The policies and policy types developed herein are only a sampling of the policies that could be generated using WASL, but are sufficient to demonstrate the expressiveness of the language. Three specific types of policy are used for demonstration purposes: Bell–LaPadula (BLP), Biba, and the Chinese Wall (CW).

Section 4.1 develops the fundamental constructs for implementing the requirements of the BLP confidentiality model, Biba’s integrity model, and the CW conflict of interest model. Section 4.2 demonstrates how to use WASL to specify policies of the various types. Hybrid policies are specified in the same manner as in Section 4.2 with rules constructed as illustrated in Section 4.3. The suitability of policies specified in WASL is discussed in Section 4.4 where compilation times and policy sizes are discussed. Conclusions are contained in Section 4.5.

4.1 BLP, Biba, and CW Rules in WASL

Translation from BLP requirements into WASL rules is straightforward. BLP discretionary authorizations for reading and writing translate to `cando(s,o,a)` statements. These BLP rules can be specified in two WASL rules.

```
-- enumerate system authorizations to read: discretionary right plus "dom" relation
cando(s, o, R) & inlevel(s, l1) & inlevel(o, l2) & levelgeq(l1, l2) => auth(s, o, R);
-- enumerate system authorizations to write: discretionary right plus "dom" relation
cando(s, o, W) & inlevel(s, l1) & inlevel(o, l2) & levelgeq(l2, l1) => auth(s, o, W);
```

It is clear from these rules that no system authorization (`auth`) can be produced without having a discretionary authorization (`cando`) as well as the proper security level relationships. Conversely, if these conditions are met, the system authorization will be added.

WASL–Biba also has two requirements, each a dual of a WASL–BLP rule above (without discretionary authorizations, however). Again no system authorization (`auth`) is generated unless the integrity levels of the subject and object are related appropriately. If, however, the integrity level relationship is as required, the system authorization will be entered into the system.

```
-- enumerate system authorizations to read
inlevel(s, l1) & inlevel(o, l2) & levelgeq(l2, l1) => auth(s, o, R);
-- enumerate system authorizations to write
inlevel(s, l1) & inlevel(o, l2) & levelgeq(l1, l2) => auth(s, o, W);
```

Specifying CW policies in WASL is more involved than BLP or Biba. WASL identifies when some combination of terms satisfies a given condition (reflecting an existential quantifier). WASL does not, however, directly provide identifying terms that satisfy a condition for a universal quantifier.

This appears in the CW condition to *write* (“for all unsanitized objects O' , S can read $O' \Rightarrow CD(O') = CD(O)$ ”).

To generate authorizations based on the universal quantifier using the WASL–CW *write* rule as an example:

1. Negate the requirement,
2. Enumerate instances meeting the negation, and
3. Use the absence of a negated instance to generate system authorizations for the related access right.

The application of this procedure considers the partial CW requirement stating that s may *write* o if, “for all unsanitized objects o' , s can read $o' \Rightarrow CD(o') = CD(o)$.” The negation can be expressed in the following equivalent statements, including the translation into WASL, that identify when s may not *write* o .

- NOT (for all unsanitized objects o' , s can read $o' \Rightarrow CD(o') = CD(o)$)
- For any unsanitized object o' , NOT (s cannot read o' OR $CD(o') = CD(o)$)
- For any unsanitized object o' , s can read o' AND $CD(o') \neq CD(o)$)
- `-in(o2,CDSan) & do(s,o2,R) & ((in(o1,k1) & dirin(o1,CD)) & (in(o2,k2) & dirin(k2,CD)) & -equals(k1,k2))`

An additional complication in implementing CW in WASL is the use of historical data to limit accesses to objects according to previous subject accesses. In a WSN, that a subject has accessed a given object may take a long time to propagate throughout the WSN. A subsequent access by that subject, while prohibited by CW rules, may be permitted because the system is not yet aware of the previous event. For this reason, WASL–CW lists permitted authorizations and prohibits all others. Initial authorizations are expressed in the policy specification as `do(s,o,a)`.

The CW *read* requirement in WASL is captured in a single rule using variable objects `o1` and `o2`, a variable kind `k1`, and a variable subject `s`.

```
-- authorized Reads
dirin(o1, CDSan) | (do(s, o2, R) & in(o1, k1) & in(o2, k1) & dirin(k1, CD))
=> auth(s, o1, R);
```

Using the same variables names as in *read* authorizations, plus the kind variable `k2`, the following WASL *write* rules enforce CW requirements.

```

-- unauthorized writes
-in(o2, CDSan) & -in(o1, CDSan)
  & ((in(o1, k1) & in(o2, k2) & dirin(k1, CD) & dirin(k2, CD) & -equals(k1, k2))
    & do(s, o2, R))
  => do(s, o1, -W);
-- authorized Writes
-do(s, o1, -W) & auth(s, o1, R)
  => auth(s, o1, W);

```

Note that the rule determining unauthorized writes is slightly different from that derived above. It will not generate a “disallowed *write*” for a “sanitized” object. It is assumed that appropriate precautions have been taken that ensure sanitized data is indeed sanitized. In addition, the last rule, $\text{-do}(s, o1, -W) \ \& \ \text{auth}(s, o1, R) \Rightarrow \text{auth}(s, o1, W)$, requires the results of the other rules prior to compilation. This policy program is, therefore, recompiled after this rule is added to the previously compiled policy program.

4.2 *Creating Policies with WASL*

This section expands on the description of the WASL rules in Section 4.1 to specify particular BLP-, Biba-, and CW-compliant policies. Section 4.2.1 develops two distinct BLP systems, Section 4.2.2 specifies a Biba system, and Section 4.2.3 specifies a CW system.

4.2.1 BLP in WASL. This section walks through the creation of the specification for system \mathbb{J} and \mathbb{K} , both consistent with the BLP definition of security. Elements of these systems are used in later sections to demonstrate hybrid confidentiality and integrity models and policy composition.

System \mathbb{J} employs a confidentiality-oriented security policy following the BLP model. It has two groups of subjects, *JGS* and *JGU* corresponding to subjects operating at security classification level *S* (*classified secret*) and *U* (*unclassified*), respectively. There is one subject in each group, *JS1* and *JS2*, respectively. A graphical depiction of the subjects and their relations is in Figure 6. A similar relationship captures two object kinds and two objects in Figure 7. Solid lines in Figures 6, 7, and 8 indicate explicitly specified relationships, dashed lines identify relationships deduced by the compiler, and arrows indicate the entities represented by a particular variable type.

The high (*JS*) and low (*JU*) sides of \mathbb{J} work together to enforce the policy. Figure 8 integrates Figures 6 and 7 to show the associations between the subjects and objects as well as the relationship between the high and low sides of the system. The one new relation, `levelorder`, associates level *JU* elements with level *JS* elements and provides the information necessary for interpreting `levelgeq`, thus implementing the *dom* relation.

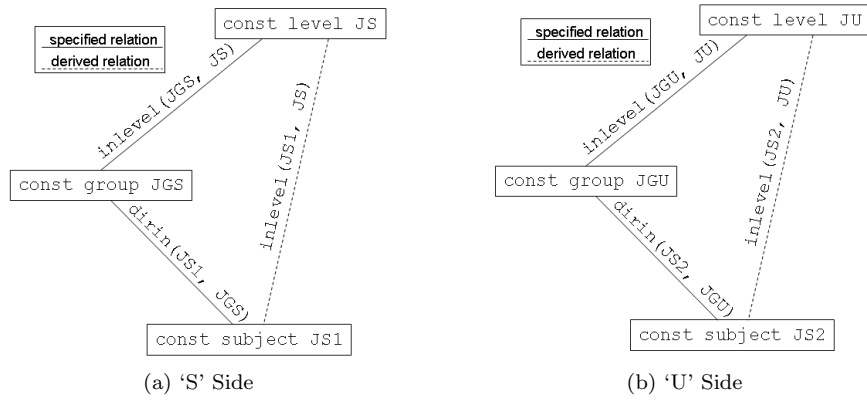


Figure 6: Subjects for \mathbb{J}

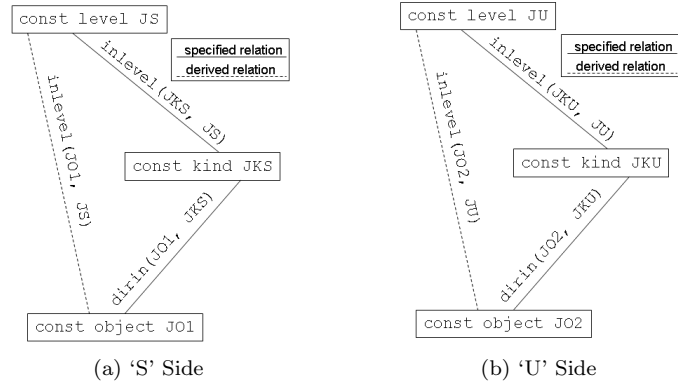


Figure 7: Objects for \mathbb{J}

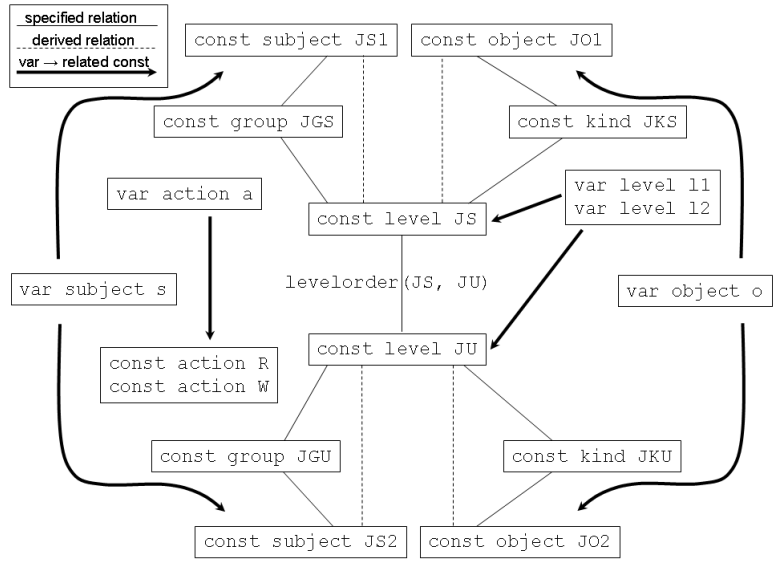


Figure 8: Structure and Components of \mathbb{J}

To complete the specification of \mathbb{J} , the code in the above figures is augmented by discretionary rights and the two rules that express BLP mandatory authorizations. The security administrator of \mathbb{J} uses a rule to generate all subject–object–action combinations as discretionary authorizations. This and the BLP rules are, with comments:

```
-- every subject has discretionary rights to all objects
true => cando(s,o,a);
-- the two BLP--enforcing rules
cando(s, o, R) & inlevel(s, 11) & inlevel(o, 12) & levelgeq(11, 12)
=> auth(s, o, R);
cando(s, o, W) & inlevel(s, 11) & inlevel(o, 12) & levelgeq(12, 11)
=> auth(s, o, W);
```

Adding the following code to the policy specification establishes a particular `leveltype` and associates the security levels with that type. This has little impact when the system operates independently of other networks, but is important for composition addressed later.

```
-- establish a leveltype and make associations
const leveltype BLP;
leveltype(JS, BLP); leveltype(JU, BLP);
```

Compiling the above code yields the discretionary authorizations below. The BLP rules specify that not all of the discretionary rights are permissible as system authorizations, so the resulting set of `auth` statements has fewer members than the set of `cando` statements.

```
cando(JS1, J01, R); cando(JS1, J01, W);
cando(JS1, J02, R); cando(JS1, J02, W);
cando(JS2, J01, R); cando(JS2, J01, W);
cando(JS2, J02, R); cando(JS2, J02, W);

auth(JS1, J01, R); auth(JS1, J01, W);
auth(JS1, J02, R); auth(JS2, J01, W);
auth(JS2, J02, R); auth(JS2, J02, W);
```

Using identical processes yields a similar system \mathbb{K} . This system has fewer discretionary authorizations. The policy code is provided with comments in Figure 9. Compiling this program adds the statements shown in Figure 10a to the original program—the `inlevel` statements generated as a result of statement analysis while the `cando` authorizations follow from the encoded rules. System authorizations presented in Figure 10b are generated by the BLP rules.

For systems \mathbb{J} and \mathbb{K} it is clear that despite any discretionary authorizations, a system authorization is not added unless the relative security levels of the subject and object comply with BLP rules. Equally clear is that system authorizations are not added unless a matching discretionary authorization is also in the system. Thus, BLP requirements are enforced by the WASL rules.

4.2.2 Biba in WASL. System \mathbb{L} has an integrity–focused policy following the Biba model of integrity. It has two groups of subjects and two kinds of objects, each corresponding respectively to

```

begin
-- declare subjects, groups, objects, kinds,
-- and actions
const subject KS1; const subject KS2;
const group KGS; const group KGU;
const object KO1; const object KO2;
const kind KKS; const kind KKU;
const action W; const action R;
-- declare levels/establish hierarchy
const level KS; const level KU;
levelorder(KS, KU);
-- establish leveltype and make associations
const leveltype BLP;
leveltype(KS, BLP); leveltype(KU, BLP);
-- relate subjects-groups and groups-levels
dirin(KS1, KGS); dirin(KS2, KGU);
inlevel(KGS, KS); inlevel(KGU, KU);
-- relate objects-kinds and kinds-levels
dirin(KO1, KKS); kindofdirin(KO2, KKU);
inlevel(KKS, KS); inlevel(KKU, KU);
-- establish a leveltype/make associations
const leveltype K;
leveltype(KS, K); leveltype(KU, K);
-- declare subject, object, action,
-- and level variables
var subject s; var object o; var action a;
var level l1; var level l2;
-- specify discretionary accesses
cando(KS1, KO2, W);
true => cando(s, KO1, R);
true => cando(KS2, KO2, a);
-- incorporate the two BLP rules
cando(s, o, R) & inlevel(u, l1)
  & inlevel(o, l2) & levelgeq(l1, l2)
  => auth(s, o, R);
cando(s, o, W) & inlevel(s, l1)
  & inlevel(o, l2) & levelgeq(l2, l1)
  => auth(s, o, W);
end;

```

Figure 9: Specification of \mathbb{K}

```

inlevel(KS1, KS); inlevel(KS2, KU);
inlevel(KO1, KS); inlevel(KO2, KU);
cando(KS1, KO1, R); cando(KS2, KO2, R);
cando(KS2, KO1, R); cando(KS2, KO2, W);

```

a: Statements Deduced During Compilation

```

auth(KS1, KO1, R);
auth(KS2, KO2, W);
auth(KS2, KO2, R);

```

b: Generated System Authorizations

Figure 10: \mathbb{K} Compilation Results

subjects and objects at integrity level H (*high integrity*) and L (*low integrity*). There is one subject in each group and one object of each kind.

Functioning as the dual of BLP-oriented systems, \mathbb{L} is constructed in a manner that correlates with \mathbb{J} as presented in Figures 6–8. Entity structures and relationships are identical but the rules differ most noticeably in the absence of discretionary authorizations. A graphical representation of the constants and relations of \mathbb{L} with the corresponding WASL code in Figure 11.

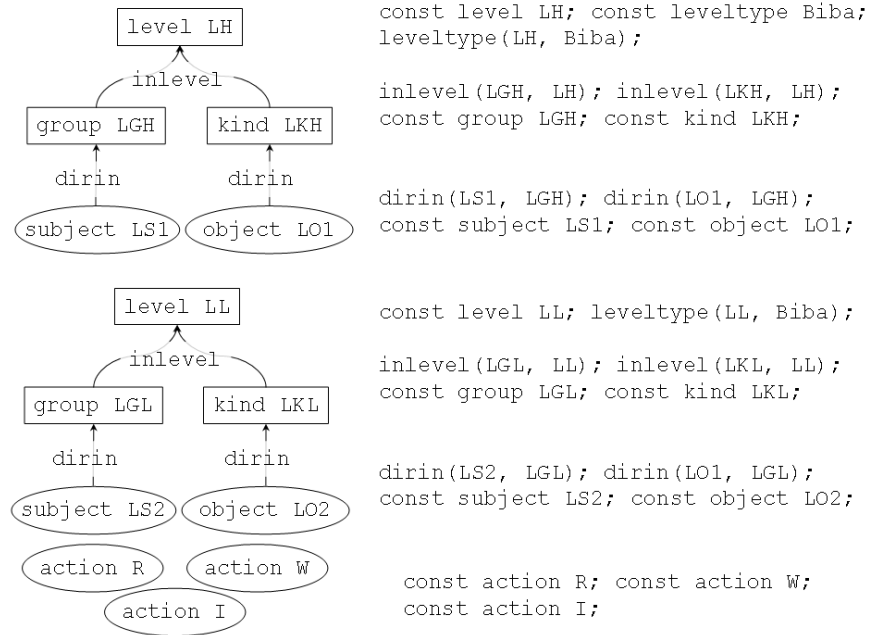


Figure 11: Construction of \mathbb{L}

Repeating the WASL rules enforcing Biba’s model, the following code is added to \mathbb{L} .

```

var subject s; var object o;
var level l1; var level l2;
inlevel(s, l1) & inlevel(o, l2) & levelgeq(l2, l1) => auth(s, o, R);
inlevel(s, l1) & inlevel(o, l2) & levelgeq(l1, l2) => auth(s, o, W);

```

Authorizations for \mathbb{L} are generated during compilation:

```

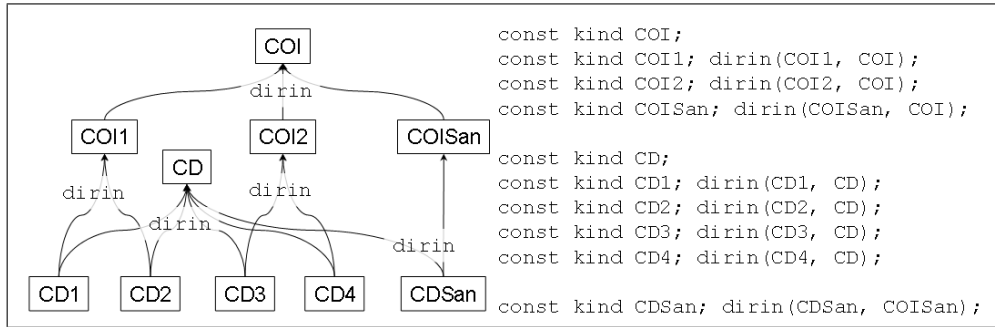
auth(LS1, LO1, R); auth(LS1, LO1, W);
auth(LS1, LO2, W); auth(LS2, LO1, R);
auth(LS2, LO2, R); auth(LS2, LO2, W);

```

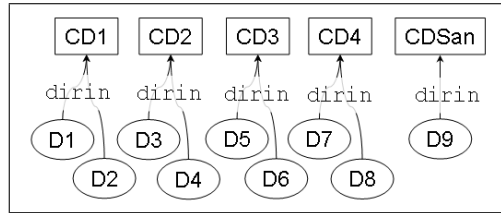
None of the subjects at the *low* level may write to objects at the *high* level and subjects at the *high* level are not authorized to read *low* level objects. This demonstrates the system maintains integrity by Biba’s definition.

4.2.3 *Chinese Wall in WASL.* This section introduces a WASL-encoded example of the Chinese Wall (CW) and discusses how a Chinese Wall security policy could be implemented in a WSN. The primary classifications for CW systems involve the categorization of data objects. Conflict of interest classes (COIs) and company datasets (CDs) compartmentalize data and so constitute two kinds in the WASL specification.

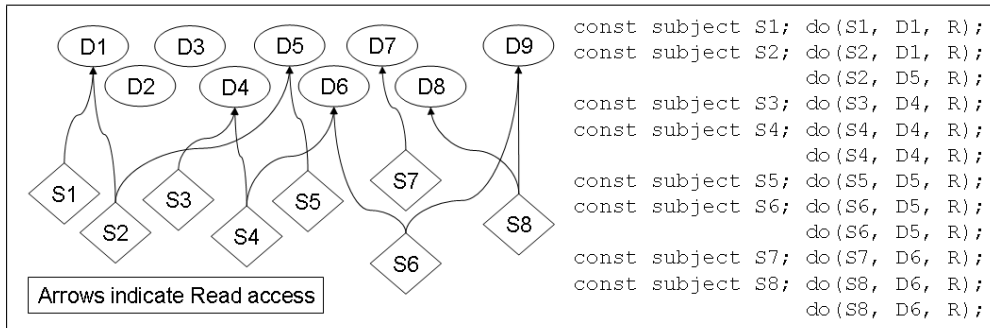
System \mathbb{P} implements the COI and CD kinds with two COIs and four datasets. Figure 12a graphically depicts these kinds and provides the corresponding WASL code. Two objects in each CD are shown with associated code in Figure 12b while eight subjects and their initial accesses are presented in Figure 12c. Sanitized data resides in the CD CDSan.



a: COIs and CDs for \mathbb{P}



b: Objects with CDs for \mathbb{P}



c: Subjects with Initial Read Accesses for \mathbb{P}

Figure 12: Initial Relationships in System \mathbb{P}

The first two CW rules,

```

-- authorized reads
dirin(o1, San) | (do(s, o2, R) & in(o1, k1) & in(o2, k1) & dirin(k1, CD))
=> auth(s, o1, R);
-- unauthorized writes
-in(o2, CDSan) & -in(o1, CDSan)
& ((in(o1, k1) & in(o2, k2) & dirin(k1, CD) & dirin(k2, CD) & -equals(k1, k2))
& do(s, o2, R))
=> do(s, o1, -W);

```

are added to the code segments in Figure 12 for compilation yielding the mandatory negative authorizations ($\text{do}(u, o, -W)$ statements), the *read* accesses in Figure 13, and system authorizations (*auth* statements) in Figure 14.

```

do(S1, D3, -W); do(S1, D4, -W); do(S1, D5, -W); do(S1, D6, -W);
do(S1, D7, -W); do(S1, D8, -W);
do(S2, D1, -W); do(S2, D2, -W); do(S2, D3, -W); do(S2, D4, -W);
do(S2, D5, -W); do(S2, D6, -W); do(S2, D7, -W); do(S2, D8, -W);
do(S3, D1, -W); do(S3, D2, -W); do(S3, D5, -W); do(S3, D6, -W);
do(S3, D7, -W); do(S3, D8, -W);
do(S4, D1, -W); do(S4, D2, -W); do(S4, D3, -W); do(S4, D4, -W);
do(S4, D5, -W); do(S4, D6, -W); do(S4, D7, -W); do(S4, D8, -W);
do(S5, D1, -W); do(S5, D2, -W); do(S5, D3, -W); do(S5, D4, -W);
do(S5, D7, -W); do(S5, D8, -W);
do(S6, D1, -W); do(S6, D2, -W); do(S6, D3, -W); do(S6, D4, -W);
do(S6, D7, -W); do(S6, D8, -W);
do(S7, D1, -W); do(S7, D2, -W); do(S7, D3, -W); do(S7, D4, -W);
do(S7, D5, -W); do(S7, D6, -W);
do(S8, D1, -W); do(S8, D2, -W); do(S8, D3, -W); do(S8, D4, -W);
do(S8, D5, -W); do(S8, D6, -W);

```

Figure 13: Derived Write Accesses in \mathbb{P}

```

auth(S1, D1, R); auth(S1, D2, R); auth(S1, D9, R);
auth(S2, D1, R); auth(S2, D2, R); auth(S2, D5, R); auth(S2, D6, R); auth(S2, D9, R);
auth(S3, D3, R); auth(S3, D4, R); auth(S3, D9, R);
auth(S4, D3, R); auth(S4, D4, R); auth(S4, D5, R); auth(S4, D6, R); auth(S4, D9, R);
auth(S5, D5, R); auth(S5, D6, R); auth(S5, D9, R);
auth(S6, D5, R); auth(S6, D6, R); auth(S6, D9, R);
auth(S7, D7, R); auth(S7, D8, R); auth(S7, D9, R);
auth(S8, D7, R); auth(S8, D8, R); auth(S8, D9, R);

```

Figure 14: System Authorizations for \mathbb{P} After First Compilation

Compilation is performed again after adding the last CW rule:

```

-do(s, o1, -W) & auth(s, o1, R) => auth(s, o1, W);

```

The authorizations resulting from this compilation include the system authorizations specifying *write* accesses in Figure 15.

Thus, no subject may read objects in multiple CDs within the same COI. Similarly, a subject is not authorized to write to an object if it has had access to an object in another unsanitized dataset. Such illegal authorizations cannot be generated due to the WASL rules enforcing CW constraints.

```

auth(S1, D1, W); auth(S1, D2, W); auth(S1, D9, W);
auth(S2, D9, W);
auth(S3, D3, W); auth(S3, D4, W); auth(S3, D9, W);
auth(S4, D9, W);
auth(S5, D5, W); auth(S5, D6, W); auth(S5, D9, W);
auth(S6, D5, W); auth(S6, D6, W); auth(S6, D9, W);
auth(S7, D7, W); auth(S7, D8, W); auth(S7, D9, W);
auth(S8, D7, W); auth(S8, D8, W); auth(S8, D9, W);

```

Figure 15: Additional System Authorizations for \mathbb{P} After Second Compilation

4.3 Rule Construction for Hybrid Policies

CW is considered a hybrid policy because it has elements reflecting the ideas of both confidentiality and integrity. Using WASL, however, the principles of multiple policy types can be expressly enforced. This section demonstrates the creation of rules for BLP–Biba and BLP–CW combinations.

Developing policy rules that enforce a hybrid BLP–Biba policy is simple because there is one rule specifying the requirements for *read* authorizations and one for *write*. The conjunction of the rules associated with the relevant action is almost all that is necessary with an exception being the explicit association of referenced security levels with either BLP or Biba.

The following are the rules for the BLP–Biba system.

```

-- enumerate system authorizations to read
leveltype(l1, BLP) & leveltype(l3, Biba) & cando(s, o, R) & inlevel(s, l1)
& inlevel(o, l2) & levelgeq(l1, l2) & inlevel(s, l3) & inlevel(o, l4)
& levelgeq(l4, l3)
=> auth(s, o, R);
-- enumerate system authorizations to write
leveltype(l1, BLP) & leveltype(l3, Biba) & cando(s, o, W) & inlevel(s, l1)
& inlevel(o, l2) & levelgeq(l2, l1) & inlevel(s, l3) & inlevel(o, l4)
& levelgeq(l3, l4)
=> auth(s, o, W);

```

The rules thus require any subjects and objects in the system to have an association with both a BLP security level and a Biba security level if they are to appear in any system authorizations. In addition, the desired relationship between the levels of both systems must be appropriately met for the system authorization to be granted.

BLP and CW can similarly be combined. Using the following WASL rules as the only means for generating *read* and *write*, system authorizations are simultaneously consistent with the BLP and CW axiomatic systems.

```

-- enumerate system authorizations to read
(cando(s, o1, R) & inlevel(s, l1) & inlevel(o1, l2) & levelgeq(l1, l2))
& (in(o1, CDSan) | (do(s, o2, R) & in(o1, k1) & in(o2, k1) & dirin(k1, CD)))
=> auth(s, o1, R);
-- enumerate mandatory negative write

```

```

-- authorizations
-in(o2, CDSan)
  & ((in(o1, k1) & in(o2, k2) & dirin(k1, CD) & dirin(k2, CD) & -equals(k1, k2))
    & do(s, o2, R))
  => do(s, o1, -W);
-- enumerate system authorizations to write
cando(s, o1, W) & inlevel(s, l1) & inlevel(o1, l2) & levelgeq(l2, l1)
  & in(o1, CD) & auth(s, o1, R) & -do(s, o1, -W)
  => auth(s, o1, W);

```

Combining Biba and CW systems is not presented as it is a trivial adaptation of this BLP–CW combination. System generation follows the same patterns as shown above for the individual BLP, Biba, and CW policy implementations with the difference being the simultaneous use of two systems. Assurance of security is again by definition and can be proven (proofs are provided in Chapter VI) for axiomatic policy types such as those discussed above.

4.4 Suitability of WASL for a WSN

This section examines implementation issues and shows it is reasonable to transmit and employ the system authorizations of composed policies within a WSN. The data in this section was generated using a Java Compiler Compiler parser generator (JavaCC™ Version 4.0) for WASL parsing and grammar checking and an abstract syntax tree for semantic checking and further manipulations. Information regarding the syntax tree methodology used can be found in [App98]. Policy composition is presented in Chapter V, but the results are presented here as well.

A WSN may be composed of hundreds, if not thousands of nodes. The number of objects in the system could be equally large, with some data available on a per–node basis. The possible actions that must be controlled by a policy are expected to be small, but a BLP policy like \mathbb{J} for 1000 nodes and 1000 objects with 6 actions, has 6,000,000 potential authorizations. If a unique binary number identifies each authorization, the representation requires nearly 3 bytes to express ($2^{22} < 6,000,000 < 2^{23}$, so 23 bits per authorization are required).

Additionally, a policy of this size would be prohibitively large to compile. Nodes are likely in one or more groups, so there would be 1000 or more statements capturing these relationships. Similarly, the 1000 objects would be of one or more kinds, leading to at least 1000 more statements. A simple rule with two variables, one of type `object` and one of type `subject`, and two relation expressions; compilation requires up to 1000×1000 term substitutions and the evaluation of the same number of expressions. As the size of the policy increases, the worst case scenario is a factorial expansion of the required compilation time. Memory required to store this information grows at a similar rate.

Representing the policy via rules at each node is an equally unappealing option, given the characteristics just described. Either the nodes would be required to store membership and other information about entity relationships or much of this information would be required for each query. The cost in memory, computational requirements, or transmission bandwidth for this is prohibitive. Separate node identifiers, however, are unlikely. More probable is a limited number of node groups and kinds of data manipulated. Using WASL to specify system authorizations by group and/or kind significantly decreases the volume of information required to determine whether or not a request is authorized.

Improvements on the proposed process of policy system authorization distribution, however, can also be made. Data compression methods, for example, decrease the bandwidth required for policy transmission even further as it reduces memory consumption at the nodes. This being so, the following alternative approach takes advantage of WSN characteristics.

Consider a network with thousands of nodes but only three subject **groups**, five object **kinds**, and four possible **actions**. There are now only sixty possible authorizations—a policy size that, using the simplistic binary system described above (one byte per authorization with 60 bytes maximum), is a reasonable size for transmission to and storage at a node.

Policy compilation speed trials have been performed on a notebook computer with 1300 MB of memory and an AMD Athlon 3400 processor running at 2.2 GHz. Each network had two **groups**, two **kinds**, two **levels**, and two **actions** with a varying number $||S||$ of **subjects** (where $||S||$ is the size of set S) and $||O||$ of **objects** uniformly distributed across **groups** and **kinds** as appropriate, with the latter two also uniformly distributed across **levels**. The two actions are *read* and *write* while the groups (subject types) and kinds (object types) correspond to *secure* and *unsecure* confidentiality levels. A **dirin** statement associates each element of S with exactly one group while another identifies objects in O as exactly one kind. The two rules identified in the examples above

```
cando(s, o, R) & inlevel(s, 11) & inlevel(o, 12)
    & levelgeq(11, 12) => auth(s, o, R);
cando(s, o, W) & inlevel(s, 11) & inlevel(o, 12)
    & levelgeq(12, 11) => auth(s, o, W);
```

specify the confidentiality requirements of the BLP model [BL75].

When using `true => cando(s,o,a)` to generate grant all discretionary accesses for the two-action system, $2 \times ||S|| \times ||O||$ term substitutions replace the action, subjects, and object variables to generate $2 \times ||S|| \times ||O||$ discretionary authorizations. Next, order $||S|| \times ||O|| \times 2 \times 2$ term substitutions replace the variables `s`, `o`, `11`, and `12` with the subjects, objects, and security levels in the system.

Execution times, however, grow at a rate much greater than $\|S\| \times \|O\|$. Table 13 contains the compilation times with the 95% confidence interval for a two-level BLP security system with 2 actions, 5 objects, and a number of subjects ranging from 10 to 1000. System authorizations are based on discretionary authorizations that provide all possible authorizations for every subject-object pairing using the rule `true => cando(s,o,a)` and the two BLP mandatory access control rules. Each compilation was run 10 times with interfering processes on the laptop kept to a minimum. Runs have also been completed for a variety of other combinations of subjects and objects including a run with 60 objects and 60 subjects requiring 233 seconds and another with 50 objects and 100 subjects requiring 533 seconds.

Table 13: 5-Object BLP Policy Compilation Times

# Subjects	Compilation Time (sec.)
10	0.03 ± 0.01
20	0.05 ± < 0.01
30	0.11 ± < 0.01
50	0.21 ± 0.01
70	0.51 ± 0.01
100	0.99 ± 0.04
200	2.01 ± 0.05
300	8.20 ± 0.08
500	20.01 ± 0.24
700	63.633 ± 0.72
1000	358.71 ± 1.75
N/A ¹	0.02 ± 0.01

¹ Authorizations are identified by group with a five groups. The number of subjects can grow without impact to the compilation time.

Nodes in a WSN are unlikely to be managed individually, with distinct authorizations per node. Groups of nodes are expected to perform similar functions throughout the WSN and authenticate by type rather than individually. To match this expectation, test runs using rules focused on groups rather than individual subjects were done. Compiling the policy for a WSN with 100,000 nodes associated with five groups, with the same number of other parameters described for Table 13, but focusing the rules on granting authorizations by group association and including only the relations required by these rules, the compilation time is 0.02 seconds as shown in the table. Because the particular subjects are not a factor given a constant number of groups, the number of nodes can be

increased or decreased without affecting the compilation time or memory required for storage of the policy.

Composition is an extension of policy compilation that is performed on the combination of two previously-compiled policies. The non-system authorizations (mandatory and discretionary authorizations) of each are removed and composition statements are added in preparation for processing. The time required for composition may be significantly less than that for compilation, depending primarily on the number of constants corresponding to the variables of the composition rules but also on the number of relations interacting with the given rules. Table 14 shows the composition time for extending low-level read access from a five-object, 10-subject system to similar five-object systems with the listed numbers of subjects. The time requirements for extending low-level read access from a five-object system (with varied numbers of subjects as listed) to a five-object, ten-subject system is shown in Table 15.

Plain text representations of a 5 *object* \times 1000 *subject* policy (including all relation statements) as described above requires 578 KB of memory when stored as plain text, but simplifying the representation and representing the authorizations by group rather than by subject quickly reduces the size to 458 bytes for the entire set of system authorizations. Assuming a given node interacts with only three of the five objects, the storage of system authorizations at a node requires only 210 bytes of memory. While these are more reasonably sized for transmission to and storage on a node, a representation using a binary code rather than text would result in further compression.

4.5 *Conclusion*

WASL is a policy language that can represent arbitrary policies and generate authorizations consistent with such security models as Bell-LaPadula, Biba, or the Chinese Wall. This new language, with its associated compiler, provides the foundation for a policy-enforcement system in which the policy may be changed without modification of the enforcement mechanisms; this capability adds to the flexibility and security of query-based WSNs.

The translation of requirements of the BLP, Biba, and Chinese Wall models into WASL code has been demonstrated and the hybrid versions of these models are straightforward. To represent a policy in WASL with the intended application in a WSN, the complete enumeration of authorizations is necessary. This means that dependence on historical data, such as is required by the Chinese Wall, must be removed. System authorizations consistent with the requirements of the Chinese Wall have been generated using an initial state and rules that prohibit authorizations depend on unknown future activities.

Table 14: Times for Composition of a 5–Object, 10–Subject System with a 5–Object, X–Subject System

# Subjects	Composition Time (sec.)
10	0.05 ± 0.01
20	0.10 ± 0.03
30	$0.11 \pm < 0.01$
50	0.20 ± 0.04
70	0.29 ± 0.03
100	0.41 ± 0.01
200	1.48 ± 0.04
300	3.46 ± 0.06
500	9.82 ± 0.12
700	18.95 ± 0.22
1000	38.69 ± 0.37

Table 15: Times for Composition of a 5–Object, X–Subject System with a 5–Object, 10–Subject System

# Subjects	Composition Time (sec.)
10	$0.06 \pm < 0.01$
20	0.17 ± 0.08
30	0.25 ± 0.08
50	0.32 ± 0.01
70	0.57 ± 0.06
100	1.00 ± 0.01
200	5.04 ± 0.06
300	11.99 ± 0.11
500	38.75 ± 0.14
700	89.88 ± 0.47
1000	179.16 ± 0.59

A straightforward implementation of a WASL system has been created to read, compile, and compose security policies. The system is naïve in that it implements the algorithms described above with few optimizations. Policies for these networks are specified in WASL and compiled. The resulting system authorizations are combined using composition rules to yield new sets of system authorizations that are compatible and consistent with the original security policies. System authorizations are compressed to a reasonable size for transmission throughout a WSN and for storage on a node.

Tests identify that policies with several objects and up to one thousand subjects/groups can be compiled and composed with other policies on a notebook-class computer. As long as the number of entities represented in the system is not excessive, the resulting policy can be reasonably transmitted to and processed at a typical WSN node. The number of entities is a crucial measure because as they increase, the time required for compilation increases factorially in the worst case while the space required for policy storage increases linearly. Given a reasonable grouping of subjects, however, policies for WSN's with hundreds of thousands or millions of nodes can be compiled and effectively implemented.

V. Composition in WASL

Using WASL as described in Chapter III and the policies developed in Chapter IV, this chapter focuses on the composition of policies using WASL. Composition is defined in Section 5.1 and then demonstrated with examples in Section 5.2.

5.1 Defining Policy Composition

Composition in WASL uses WASL’s compilation procedures to merge multiple policies by manipulating the policies and adding rules that provide cross-policy interactions. The concepts presented here parallel some of those found in [GQ94] and [BdVS02].

Suppose there are independent systems A and B , in which subjects from one system need to interact with objects from the other; policy composition ensures the security policies defined individually within A and B are maintained while permitting additional authorizations so subjects in A access objects in B and/or vice-versa. In such a system both the authorizations and prohibitions in effect under isolated systems A and B must be preserved. A system that meets these initial requirements is referred to as being *compatible* with the original. Additionally, the interaction of A with B should be consistent, not leading to direct or indirect violations of either of the policies.

The *compatibility* requirement is partially met by a composition when all authorizations in a policy are also in the composition’s resultant system authorizations. When a composition meets this condition and, additionally, when every action that is disallowed by a policy is also disallowed in the composition, compatibility is fully satisfied. Both of these conditions are met by requiring any additions to system authorizations to have actors and targets from separate systems. Thus, changes to a given system’s policy cannot be accomplished via composition. *Consistency* is achieved when data flows prohibited in the individual policies are also prohibited in the composition. Direct violations are addressed by the correctness description above. Indirect violations are prohibited by ensuring a partial order of security levels between systems is established and maintained.

Consider the following example of an indirect violation. Suppose a subject s under policy A is not permitted *read* access to object o (i.e., either the expression $\text{-auth}(s,o,\text{read})$ or $\text{auth}(s,o,\text{-read})$ is **true**) as presented in Figure 16. After composition with policy B , s_1 (under policy B) is authorized to *read* o ($\text{auth}(s_1,o,\text{read})$) and *write* o_1 ($\text{auth}(s_1,o_1,\text{write})$). If s is authorized to *read* o_1 ($\text{auth}(s,o_1,\text{read})$), this indirectly violates the policy for system A since s could read o data written to o_1 by s_1 . This situation cannot occur when security levels are properly ordered during composition.

This leads to the definition of secure composition adopted here, and reflect those identified in [GQ94]:

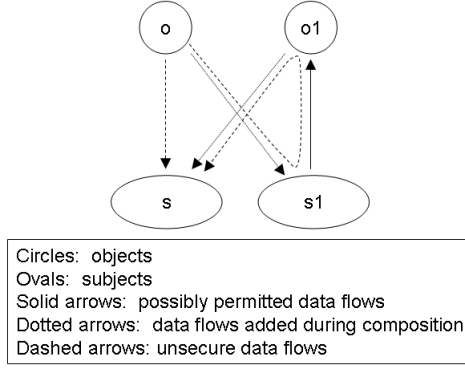


Figure 16: Unsecure Composition

Secure Composition: A composition of two security policies A and B , yielding a security policy C is secure when the following hold simultaneously:

1. System authorizations for subjects under policy A (B) are unchanged in policy C with respect to objects under policy A (B).
2. Authorizations in C not appearing in either A or B comply with the policy restrictions of both policies A and B .

For an effective implementation in WASL, policies to be merged must include the system authorizations, the other non-authorization relations, and the declarations of any terms included in these authorizations. Rules generating authorizations during compilation must be removed during composition, however, as they are useful only during compilation and could lead to undesired effects during composition.

Consider an example of just such an undesired effect. Suppose the intent of a composition of systems \mathbb{J} and \mathbb{K} is to permit subjects in \mathbb{K} at level KS to read objects in \mathbb{J} at level JU , but to prohibit any access for subjects in \mathbb{K} at level KU . If the security administrator for \mathbb{J} used the rule $R_{orig} = \text{true} \Rightarrow \text{cando}(s, o, a)$ to generate the discretionary authorizations under a BLP policy and this rule is not removed, the composition rules will permit the accesses expressly prohibited if the intended discretionary accesses are $R_{new} = \text{in}(s, \text{KS}) \ \& \ \text{in}(o, \text{KU}) \Rightarrow \text{cando}(s, o, R)$. R_{orig} will generate an undesired discretionary authorization for every subject in \mathbb{K} if not removed prior to compilation.

The rules used for composition must maintain the security of the component policies. The only way to ensure this is to require consistency of the system authorization-generating rules with the component policy system authorization-generating rules. If the component policies implement differing definitions of security a hybrid set of rules is required (cf. Section 4.3).

The requirements for composition are:

1. Actors and targets in system authorizations generated by composition rules must come from different systems.
2. Terms between policies must be unambiguous (i.e., a given term identifier must represent exactly one term).
3. Except for system authorizations, authorization statements associated with one or the other of the policies to be composed must not play a role in composition.
4. Composition rules generating system authorizations must be the same as those used for the composition of the individual policies.

The steps to perform a composition that permit subjects in \mathbb{A} access to objects in \mathbb{B} are below. When each of the required associations and rules are defined in advance, these steps are easily automated.

Steps to Perform Secure Composition

1. Combine the specifications of \mathbb{A} and \mathbb{B} . Call it $\mathbb{A}\mathbb{B}$.
2. Remove all rules from $\mathbb{A}\mathbb{B}$ (formally, this is composition $\mathbb{A} + \mathbb{B}$).
3. Add necessary associations of subjects in \mathbb{A} with security elements in \mathbb{B} .
4. Add necessary associations of objects in \mathbb{B} with security elements in \mathbb{A} .
5. Add elements to generate the desired new system authorizations.
6. Add hybrid rules (rules enforcing \mathbb{A} and \mathbb{B} simultaneously) (formally, this is composition $(\mathbb{A} + \mathbb{B} * R)$).
7. Compile $\mathbb{A}\mathbb{B}$.

WASL policy composition applies elements of an algebraic framework for composing access control policies similar to those in [BdVS02]. The method permits independent policies to maintain access restrictions for systems within an environment of heterogeneous policies while adding a combined, inter-policy access specification.

In this framework policy identifiers P_x are distinguished by their subscript x and are associated with sets of relations including system authorizations through the partial mapping established by the environment e . The mapping of system authorizations and other non-authorization relations associated with P_x are identified by the annotation $[[P_x]]_e$. The algebraic operators include addition (+) and closure (*) using a set of rules and other statements R , each summarized in Table 16. The

closure function generates all permitted authorization terms by applying a set of inference rules R to a policy.

Table 16: Composition Operators, Symbols, and Semantics

Name	Smb	Definition
Addition	+	$[[P_1 + P_2]]_e \stackrel{def}{=} [[P_1]]_e \cup [[P_2]]_e$
Closure	*	$[[P * R]]_e \stackrel{def}{=} \text{closure}(R, [[P]]_e)$

There are two primary distinctions between the semantics herein and those used in [BdVS02]:

1. The environment e includes nearly the entire policy specification (except the rules) rather than just the system authorizations.
2. R can include both relations and rules rather than just rules.

Applying this composition framework to a WASL-specified system leads to the representation of compositions using the form $(P_1 + P_2) * R$ or $\text{closure}(R, ([[P_1]]_e \cup [[P_2]]_e))$. An immediate requirement following from the addition of policies is terms must be unambiguous; identifiers used for subjects, objects, or other elements under policy P_1 must not be confused with elements under policy P_2 . As mentioned before, compatibility requires that any new authorizations generated by rules and other relations identified in R must include actors and targets from different systems (i.e., if the actor is under P_1 the target must be under P_2 and vice versa). The addition of relations specifying an acceptable (partially ordered) security level hierarchy along with rules that enforce the appropriate policy ensures consistency.

Thus, the annotation $[[P_x]]_e$ of a policy that is to be compiled represents a slightly smaller set of statements than the entire representation of policy P_x . Rules generating system authorizations during compilation are consistent with the component policies and are applied during composition to ensure consistency of the resulting policy.

5.2 Composition Examples

This section considers systems \mathbb{J} , \mathbb{K} , \mathbb{L} , and \mathbb{P} , using the steps and rules from Section 5.1 to compose these systems in WASL. The results of composition sustain security as defined above. Composition of \mathbb{J} with \mathbb{L} is presented in Section 5.2.1 while \mathbb{K} is composed with \mathbb{P} in Section 5.2.2. Composition within similar security models is presented with the associated formalisms in Chapter VI.

5.2.1 *BLP–Biba Composition.* Suppose the administrator of \mathbb{J} permits *read* access of level *JU* objects to level *LL* subjects under \mathbb{L} . Steps 1 and 2 need not be presented here. Steps 3 and 4 are accomplished by asserting that appropriate \mathbb{L} subjects are in level *JU* and \mathbb{J} objects are in level *LL*. Step 5 is policy–type specific, here requiring the administrator establish discretionary *read* access for \mathbb{J} subjects. Step 6 applies the hybrid BLP–Biba policy rule that generates *read* system accesses. The code is:

```
-- associate L subjects with level JU
-- and J objects with LL
inlevel(s, LL) => inlevel(s, JU);
inlevel(o, JU) => inlevel(o, LL);
-- discretionary read permissions
-- for L subjects in level JU
inlevel(s, l1) & leveltype(l1, L) & inlevel(s, JU) & inlevel(o, l2) & leveltype(l2, J)
=> cando(s, o, R);
leveltype(l1, J) & leveltype(l3, L) & cando(s, o, R) & inlevel(s, l1) & inlevel(o, l2)
& levelgeq(l1, l2) & inlevel(s, l3) & inlevel(o, l4) & levelgeq(l4, l3)
=> auth(s, o, R);
```

Compilation (step 7) yields two discretionary authorizations but only one new system authorization:

```
cando(LS2, J01, R);
cando(LS2, J02, R);
auth(LS2, J02, R);
```

The composition permits all the accesses originally provided by \mathbb{J} and \mathbb{L} in their respective networks and does not add accesses that are disallowed in the separate systems, thus preserving security within the individual networks. The added authorization also sustains security for the composed system ($\mathbb{J}\mathbb{L}$) by permitting only *R* access to objects at the low security level in \mathbb{J} by subjects in \mathbb{L} , an authorization consistent with both the BLP and Biba models of security.

5.2.2 *BLP–CW Composition.* Consider a composition that combines systems \mathbb{K} and \mathbb{P} . Suppose the administrator of \mathbb{P} grants access to \mathbb{P} objects to subjects of \mathbb{K} . Desired accesses are generated by following the following procedures.

The first two steps are clear and require no further discussion. Subject *KS1* is granted initial access to *D2* and *D6* while subject *KS2* is given access to *D1*, *D7*, and *D8*, thus accomplishing step 3. For step 4, objects *D1*, *D3*, *D5*, and *D7* are classified as *secret*, while *D2*, *D4*, *D6*, *D8*, and *D9* are *unclassified*. All cross–system discretionary accesses (\mathbb{K} subjects and \mathbb{P} objects) are specified with a single rule for step 5. The code performing steps 3 through 6 (using the hybrid rules developed in Section 4.3 for step 6) is

```
-- relate objects to K levels
inlevel(D1, KS); inlevel(D3, KS);
```

```

inlevel(D5, KS); inlevel(D7, KS);
inlevel(D2, KU); inlevel(D4, KU);
inlevel(D6, KU); inlevel(D8, KU);
inlevel(D9, KU);
-- define initial accesses
do(KS1, D2, R); do(KS1, D6, R);
do(KS2, D1, R); do(KS2, D7, R);
do(KS2, D8, R);
-- discretionary read permissions
-- for K subjects of N objects
inlevel(s, l1) & leveltype(l1, BLP)
& in(o, CD) => cando(s, o, a);
-- add hybrid rules for first compilation
-- enumerate system authorizations to read
(cando(s, o1, R) & inlevel(s, l1) & inlevel(o1, l2) & levelgeq(l1, l2))
& (in(o1, CDSan) | (do(s, o2, R) & in(o1, k1) & in(o2, k1) & in(k1, CD)))
=> auth(s, o1, R);
-- enumerate negative write authorizations
-in(o2, CDSan)
& ((in(o1, k1) & in(o2, k2) & dirin(k1, CD) & dirin(k2, CD) & -equals(k1, k2))
& do(s, o2, R))
=> do(s, o1, -W);

```

The compilation of this code (step 7) yields the *read* accesses and negative *write* authorizations in Figure 17. Adding the remaining hybrid rule below and compiling the code again generates the *write* authorizations, but because all write authorizations appear in a $\text{do}(\text{KS}_x, \text{D}_y, -\text{W})$ statement, this does not add any authorizations (or otherwise change) the compiled, composed policy. This repeats steps 6 and 7 based on requirements unique to and consistent with the CW policy type.

```

-- enumerate system authorizations to write
cando(s, o1, W) & inlevel(s, l1) & inlevel(o1, l2) & levelgeq(l2, l1)
& in(o1, CD) & auth(s, o1, R) & -do(s, o1, -W)
=> auth(s, o1, W);

```

<pre> auth(KS1, D1, R); auth(KS1, D2, R); auth(KS1, D5, R); auth(KS1, D6, R); auth(KS1, D9, R); auth(KS2, D2, R); auth(KS2, D8, R); auth(KS2, D9, R); do(KS1, D1, -W); do(KS1, D2, -W); do(KS1, D3, -W); do(KS1, D4, -W); do(KS1, D5, -W); do(KS1, D6, -W); do(KS1, D7, -W); do(KS1, D8, -W); do(KS1, D9, -W); do(KS2, D1, -W); do(KS2, D2, -W); do(KS2, D3, -W); do(KS2, D4, -W); do(KS2, D5, -W); do(KS2, D6, -W); do(KS2, D7, -W); do(KS2, D8, -W); do(KS2, D9, -W); </pre>

Figure 17: Code for \mathbb{P} Subjects' Accesses

This composition permits all the accesses originally provided by \mathbb{K} and \mathbb{P} in their respective networks and does not add accesses that are disallowed in the separate systems, thus preserving security within the individual networks. The added authorizations also sustain security for the composed system ($\mathbb{K}\mathbb{P}$) as no violations of either the CW or BLP rules are permitted, directly or indirectly.

5.3 *Conclusion*

Composition presents various challenges depending on the definition selected. The terms *consistency* and *compatibility* emphasize the key issues of primary concern in this work, capturing the importance of data flow security in this policy enforcement approach.

The composition approach presented here is demonstrated to be secure and exhibits several desirable qualities. Support for heterogeneous policies is clear. The expressiveness of WASL supports the composition of a variety of component policy models. Formal semantics can be attached to the processes and is discussed in Chapter VI.

VI. WASL: The Formal Model

Proving secure communications within a computer network is a challenge that requires some formal analysis of the system. This chapter deals with this challenge from the perspective of specified policy requirements and the expression of those requirements in WASL. Policies expressed herein are based on three formal models of security and are identified using both formal theoretical representations and equivalent WASL constructs. The axiomatic basis of each policy model is imperative to proving security in the systems described.

The chapter is organized as follows. WASL is shown to support three distinct notions of security: the Bell–LaPadula model with the WASL representation in Section 6.1, the Biba strict integrity model and its representation in WASL in Section 6.2, and the Chinese Wall model in Section 6.3. Section 6.4 provides concluding comments.

6.1 BLP Model Formalized

This section considers Bell–LaPadula Model (BLP model or just BLP) [LB73; BL75] as specified by WASL. It begins in Section 6.1.1 with a detailed presentation of the formal model along with comparable formalizations of the model in WASL. This is followed in Section 6.1.2 by an elaboration of the composition WASL policies implementing BLP.

6.1.1 Formalisms of BLP. This section presents the BLP model and its representation in WASL beginning with a discussion in Section 6.1.1.1 of notation and the comparable WASL code. Section 6.1.1.2 defines the model and Section 6.1.1.3 adds theorems demonstrating the security of the model itself and as implemented in WASL. The discussion about BLP model notations, definitions, and theorems generally follows the presentations of [LB73] and [Bis03]. These works present two additional actions, namely, execute (meaning an action that includes neither read nor write) and the combination read–write (where information flows both toward and away from the actor in the same action). Analyzing the absence of data flow (e.g., the action execute) adds nothing to this research and is, therefore, omitted. Similarly, a directive in a WSN is expected to be either read or write and, thus, the combined read–write is also omitted.

6.1.1.1 The System: Notations. Symbols, their semantics, and the equivalent WASL representation are summarized in Table 17. Individual entities are represented by a lowercase letter and, when a set is used, the corresponding uppercase letter represents “the set of all elements” of the given type (e.g., S is a set of all subjects s). \mathbb{N} represents the positive integers, and the empty set is \emptyset .

Table 17: Symbols, Semantics, and WASL Encoding for BLP

Sym	Description	WASL Representation
a	An access right	<code>var action a or const action A</code>
b	Authorizations $b \in P(S \times O \times A)$	<code>auth(s,o,a)</code>
c	A classification level	<code>var level l or const level L</code>
d	An outcome, $d \in \{\underline{e}, \underline{n}, \underline{y}\}$	—
\underline{e}	Outcome “illegal” or “error”	—
f	Pairs (f_S, f_O)	—
f_S	A function that associates each subject with its security level	Set of <code>inlevel(s,l)</code>
f_O	A function that associates each object with its security level	Set of <code>inlevel(o,l)</code>
m	Access control matrix (discretionary rights)	Set of <code>cando</code> statements
\underline{n}	Outcome “no”	—
o	An object	<code>const object O, const kind K, var object o, or var kind k</code>
\underline{r}	Access right (action) “read”	<code>const action R</code>
r	A request for access	—
s	A subject	<code>const subject S, const group G, var subject s, or var group g</code>
v	A state; a triple (b, m, f)	—
w	An action in the system, $w \in R \times D \times V \times V$	—
\underline{w}	Access right (action) “write”	<code>const action W</code>
\underline{y}	Outcome “yes”	—
$\underline{\Sigma}$	A security system	<code>begin {statements} end;</code>

$R^{\mathbb{N}}$ is a special notation indicating the set of sequences of the elements in R (in this case, the set of sequences of access requests). One of these sequences is represented by R , where the t th element of R is r_t . When two elements are sequentially related, the latter element will be annotated with a tick ($'$), such as f' , whereas the former will have no annotation, as in f .

Elements referring specifically to constructs in WASL are presented in *Courier* font. Constant terms in WASL generally begin with a capital letter (e.g., `const action W`) whereas variable terms begin with a lowercase letter (e.g., `var action a`).

WASL code segments require term identifiers. Except as otherwise declared, the lowercase WASL terms below are variables representative of any of the pertinent identifier type.

S is the set of a system’s set of actors s , called subjects, and are encoded in WASL `const subject S` and `const group G` statements. The set of objects, or the entities acted upon, is O and is captured in `const object O` and `const kind K`. The set of actions performed by a subject on an object are labeled A ; the two actions are coded as `const action W` and `const action R`.

Subjects and objects hold a security clearance level, `const level L`, corresponding to one of the security clearances c . The functions f_S and f_O associate each subject and object, respectively, with their security clearance (a term synonymous here with security classification) and f is a pair (f_S, f_O) . The security clearances are partially ordered so that when $l_i, l_j \in L$, using `levelorder(li,lj)` or `levelorder(lj,li)`, exactly one of the following holds:

1. $l_i > l_j$,
2. $l_i = l_j$, or
3. $l_i < l_j$.

Authorizations are expressed as triples (s, o, a) with three forms of authorization defined: discretionary, mandatory, and system authorizations. Let m be a discretionary rights access control matrix established by the security administrator. Each entry in m is encoded `cando(s,o,a)`. Mandatory authorizations are specified via rules and are represented through model definitions rather than being expressed by a particular language element. System authorizations b are specified using the `auth(s,o,a)` construct. They are generated by WASL rules to enforce the mandatory authorization requirements and appear as `re => auth(s,o,a)` where `re` is a relation expression. A particular element of b is identified by b_i where i identifies the i th element of b .

The system state v is represented by the triple (b, m, f) . The sequence of requests for access R and the set of possible outcomes resulting from requests D are combined to form a sequence of actions in the system $W \subseteq R \times D \times V \times V$. The requests r are not modeled in WASL because they are handled by the system-specific query-handling mechanism. An outcome d for a request can be y (“yes”), n (“no”) or e (“illegal” or “error”). The system Σ in BLP consists of its requests, outcomes, and actions performed starting from an initial state z_0 and is annotated $\Sigma(R, D, W, z_0)$.

Using these definitions the system $\Sigma(R, D, W, z_0)$ is in state v_{t-1} and receiving a request r_i from some subject, leading to decision d_i being made by the system. This sequence of activity is an action w that describes Σ “in operation,” so to speak, moving from state v_{t-1} to state v_t due to r_i and subsequent decision d_i based on state v_{t-1} .

An appearance of *Sigma* is simply a triple describing Σ “at rest” in state v_i after request r_i and decision d_i . System Σ consists of R , D , and W having an initial state of z_0 is represented as $\Sigma(R, D, W, z_0) \subseteq X \times Y \times Z$. By letting $X = R^{\mathbb{N}}$, $Y = D^{\mathbb{N}}$, and $Z = V^{\mathbb{N}}$, then at some time $t \in \mathbb{N}$, the system is in state z_{t-1} and receives some subject’s request x_t leading to decision y_t and a transition to the (possibly different) state z_t . Thus, $(x, y, z) \in \Sigma(R, D, W, z_0)$ if and only if $\forall t \in \mathbb{N} : (x_t, y_t, z_{t-1}, z_t) \in W$. This leads to the definition of an *action*.

Definition 6.1.1. $(r, d, v, v') \in R \times D \times V \times V$ is called an action of $\Sigma(R, D, W, z_0)$ if and only if there is an $(x, y, z) \in \Sigma(R, D, W, z_0)$ and a $t \in \mathbb{N}$ such that $(r, d, v, v') = (z_t, y_t, z_{t-1}, z_t)$.

When a query (request) is presented to a network, the query-handling mechanism formats the query appropriately and, if the query matches an `auth(s, o, a)` statement generated by the WASL compiler, the decision is `y`; otherwise it is `n`. Decisions are not explicitly represented because any outcome is evident by whether the system responds to the request or not.

Certain aspects presented herein do *not* appear or are modified from LaPadula's presentation. Categories have been omitted from this discussion because this level of complexity is not expected for a wireless sensor network, but extending WASL to represent categories is straightforward, making use of the `leveltype` construct and additional `inlevel` statements that would be referenced in rules generating authorizations. Additionally, a WSN's nodes are not expected to be changing security levels so f_S is the subject's security level rather than its maximum security level and f_C , the current security level, is not used. This eliminates an element from each f .

6.1.1.2 BLP Model: Formal Specification. This section uses the above conventions, definitions, and descriptions to formally define the BLP model following [LB73] and [Bis03]. The properties identified here are used in Section 6.1.1.3 to identify a secure system (see Section 2.2.2).

WASL captures Definitions 2.2.2–2.2.4 in the two rules:

1. `cando(s, o, R) & inlevel(s, l1) & inlevel(o, l2) & levelgeq(l1, l2)`
`=> auth(s, o, R), and`
2. `cando(s, o, W) & inlevel(s, l1) & inlevel(o, l2) & levelgeq(l2, l1)`
`=> auth(s, o, W).`

The first rule enforces the *ds-property* in the requirement `cando(s, o, R)` and the mandatory conditions of the *simple security condition* and **-property* with `inlevel(s, l1) & inlevel(o, l2) & levelgeq(l1, l2)`. When an authorization `auth(s, o, R)` for a particular subject `s` to perform action `R` (read) on object `o` meets these conditions, the authorization can be added to the policy. The second rule similarly enforces the requirements for generating write authorizations.

These WASL rules use an important security relationship supported by WASL: the security level hierarchy. Using specified `levels` with defined `levelorders`, a partial ordering of classification levels is established and enforced during compilation by assuring the desired reflexive, transitive, and antisymmetric relationships. The WASL relation `levelgeq(l1, l2)` corresponds to the BLP relation $l1 \text{ dom } l2$ for classification levels $l1$ and $l2$. Associations of subjects and objects to security

clearances correspond to `inlevel` statements and, along with `levelgeq` expressions, enable direct BLP-to-WASL specifications.

A system is considered secure if every reachable state satisfies the simple security condition, the `*`-property, and the discretionary security property. For WASL this means that every authorization `auth(s, o, a)` must meet the requirements in the above two rules.

6.1.1.3 Security Theorems. The definitions in the previous section provide an axiomatic basis for making assertions about the security of a given system. This section examines theorems to prove security of systems in theory and as implemented in WASL.

Theorem 6.1.1 uses Definitions 2.2.2–2.2.4 to establish the initially secure state of some WASL system σ that implements BLP as described above.

The initial state for a WASL policy is the pre-compiled specification. Thus, there are no `auth(s, o, a)` statements as these are generated during compilation.

Theorem 6.1.1. *The initial state of WASL system σ , call it $z_{\sigma 0}$, satisfies the simple security condition, the `*`-property, and the `ds`-property.*

Proof. Initially, σ contains no system authorizations—no `auth` statements. Therefore, $b = \emptyset$ satisfying Definitions 2.2.2–2.2.4 trivially. \square

The next three theorems (Theorems 6.1.2–6.1.4) further specify the security of a state generated through the addition of a new access right.

Theorem 6.1.2. *Let $v = (b, m, f)$ satisfy the simple security condition. Let $(s, o, a) \notin b$, $b' = b \cup \{(s, o, a)\}$, and $v' = (b', m, f)$. Then v' satisfies the simple security condition if and only if either of the following conditions is true.*

1. $a = \underline{w}$.
2. $a = \underline{r}$ and $f_S(s) \text{ dom } f_O(o)$.

Proof. (1) follows from Definition 2.2.2 and v' satisfying `ssc rel f`. For (2), if v' satisfies the simple security condition, then, by definition, $f_S(s) \text{ dom } f_O(o)$ when $a = \underline{r}$. Moreover, if $f_S(s) \text{ dom } f_O(o)$, then $(s, o, a) \in b'$ satisfies `ssc rel f`. Hence, v' satisfies the simple security condition. \square

Theorem 6.1.3. *Let $v = (b, m, f)$ satisfy the `*`-property. Let $(s, o, a) \notin b$, $b' = b \cup \{(s, o, a)\}$, and $v' = (b', m, f)$. Then v' satisfies the `*`-property if and only if one of the following conditions holds:*

1. $a = \underline{w}$ and $f_O(o) \text{ dom } f_S(s)$

2. $a = \underline{r}$ and $f_S(s) \text{ dom } f_O(o)$

Proof. The theorem follows from Definition 2.2.3 and v' satisfying the $*$ -property. Conversely, by Definition 2.2.3, $\{(s, o, a)\}$ satisfies the $*$ -property. Thus, v' satisfies the $*$ -property. \square

Theorem 6.1.4. *Let $v = (b, m, f)$ satisfy the ds-property. Let $(s, o, a) \notin b$, $b' = b \cup \{(s, o, a)\}$, and $v' = (b', m, f)$. Then v' satisfies the ds-property if and only if $a \in m[s, o]$.*

Proof. If v' satisfies the ds-property, then the claim follows immediately from Definition 2.2.4. Conversely, assume that $a \in m[s, o]$. Because $(s, o, a) \in b'$, the ds-property holds for v' . Thus, v' satisfies the ds-property. \square

The security state of a WSN using WASL does not change unless there is a policy update across the entire network as directed by the gateway node. Therefore, there are no state changes resulting from any request or outcome in the BLP formal model. The security state v of the WSN's security system is determined from the `auth` statements (representing b) generated by rules operating on `cando` statements (discretionary authorizations identified in m) and classification level comparisons.

The above principles extend Theorem 6.1.5 for WASL which show WASL system σ conforms to the conditions in Theorems 6.1.2–6.1.4.

Theorem 6.1.5. *For WASL system σ , only secure state changes occur during compilation.*

Proof. Definition 2.2.1 and the semantics of the relations `levelorder` and `levelgeq` establish that security levels are partially ordered.

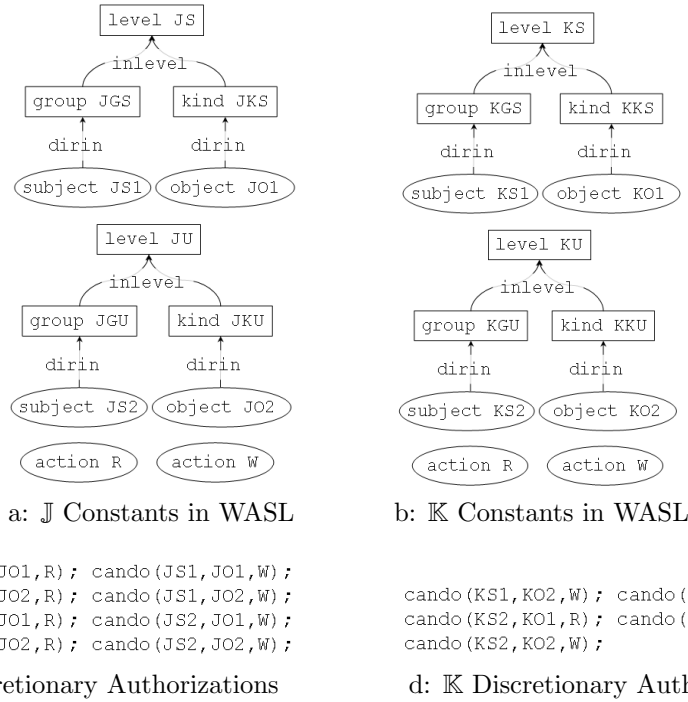
The only way to add a system authorization (s, o, \underline{r}) for some u and o , (i.e., `auth(s,o,R)`) is for $f_S(s) \text{ dom } f_O(o)$ for the discretionary authorization in m (a statement `cando(s,o,W)`). The WASL rule `cando(s,o,R) & inlevel(s,l1) & inlevel(o,l2) & levelgeq(l1,l2) => auth(s,o,R)` is the only way to add a system authorization for \underline{r} access and $\{s : s \in S\}$, thus meeting the requirements of Theorems 6.1.2–6.1.4.

The only way to add a system authorization (s, o, \underline{w}) for some s and o , (`auth(s,o,W)` in WASL), where $s \in S$ and $o \in O$ is for $f_O(o) \text{ dom } f_S(s)$ for the discretionary authorization in m (a statement `cando(s,o,W)`). The WASL rule `cando(s,o,W) & inlevel(s,l1) & inlevel(o,l2) & levelgeq(l2,l1) => auth(s,o,W)` is the only way to add a system authorization for \underline{w} access and $\{s : s \in S\}$, thus meeting the requirements of Theorems 6.1.2–6.1.4.

Other policy modifications can be made (e.g., the addition of `inlevel` statements), but they do not change the security state of the system. Therefore, state transitions that occur during compilation are secure and \mathbb{J} remains secure during compilation. \square

6.1.2 *Composition of BLP Systems in WASL.* Secure policy composition is also supported in WASL. Consider two individual policies, \mathbb{J} and \mathbb{K} . These policies are composed and the resulting policy is shown to maintain security.

System \mathbb{J} has two security levels, two groups, one subject per group, two kinds, one object per kind, and the two actions \mathbf{R} and \mathbf{W} . Figures 18a and 18c depict the specification of constants in the system and their relationships as well as discretionary authorizations defined by the system administrator. Constants in ovals are used in system authorizations while those that are not are in boxes. Key terms identifying relations relating two constants are written across the lines that identify the relationship. The specification for system \mathbb{K} is similar, appearing in Figures 18b and 18d. This policy has more limited discretionary authorizations than \mathbb{J} .



```
cando(JS1,JO1,R); cando(JS1,JO1,W);
cando(JS1,JO2,R); cando(JS1,JO2,W);
cando(JS2,JO1,R); cando(JS2,JO1,W);
cando(JS2,JO2,R); cando(JS2,JO2,W);
```

```
cando(KS1,KO2,W); cando(KS1,KO1,R);
cando(KS2,KO1,R); cando(KS2,KO2,R);
cando(KS2,KO2,W);
```

Figure 18: BLP Systems \mathbb{J} and \mathbb{K} Specifications

The policy specifications include two BLP rules, one each for generating system authorizations for Reads and Writes: $\text{cando}(s,o,R) \ \& \ \text{inlevel}(s,l_1) \ \& \ \text{inlevel}(o,l_2) \ \& \ \text{levelgeq}(l_1,l_2) \Rightarrow \text{auth}(s,o,R)$ and $\text{cando}(s,o,W) \ \& \ \text{inlevel}(s,l_1) \ \& \ \text{inlevel}(o,l_2) \ \& \ \text{levelgeq}(l_2,l_1) \Rightarrow \text{auth}(s,o,W)$. Compilation of the programs yield system authorizations listed in Figure 19. Thus, \mathbb{J} and \mathbb{K} are instances of a BLP-compliant policies in WASL by Theorems 6.1.1 and 6.1.5.

Secure composition presumes the two systems being composed share a compatible definition of security in the component policies (e.g., BLP). The composed policy must also maintain the

```

auth(JS1, J01, R); auth(JS1, J01, W);
auth(JS1, J02, R); auth(JS2, J01, W);
auth(JS2, J02, R); auth(JS2, J02, W);

```

a: \mathbb{J} System Authorizations

```

auth(KS1, K01, R);
auth(KS2, K02, R); auth(KS2, K02, W);

```

b: \mathbb{K} System Authorizations

Figure 19: BLP System Authorizations Generated by Compilation

integrity of component policies by neither removing nor adding system authorizations for subjects with objects in either system. Thus, any added system authorizations must be for subjects in one system to objects in the other.

Additionally, the WASL rules generating system authorizations must be identical to those used by the systems that are to be composed. In the case of BLP systems, it is imperative that rules generating discretionary authorizations be removed prior to composition because failure to do so could lead to the addition of unintended discretionary authorizations either within or between systems. For example, if the security administrators of \mathbb{J} and \mathbb{K} intend to use a small subset of the possible discretionary *read* authorizations for composition, but \mathbb{J} uses rule `true => cando(s,o,R)` to generate discretionary authorizations, the intent of the administrators will be thwarted by the pre-existing rule.

The following example is a policy composition along with a proof that the composition is secure. Suppose the \mathbb{J} and \mathbb{K} security administrators permit \mathbb{K} subjects access to \mathbb{J} objects and determine security level KU is equivalent to JU . Generating the composed policy begins by combining the two previously-compiled policy specifications and removing any discretionary authorization-generating rules. This step is easily automated. Two rules are added: one to associate \mathbb{K} subjects at level KU with level JU and the other to create discretionary authorizations for \mathbb{K} subjects with \mathbb{J} objects.

```

-- relate JU to KU
inlevel(s,KU) => inlevel(s,JU);
-- generate discretionary authorizations
inlevel(s,11) & inlevel(o,12) & leveltype(11,K) & leveltype(12,J) => cando(s,o,a);

```

Theorem 6.1.6 establishes the security of $\mathbb{J}\mathbb{K}$ at this point.

Theorem 6.1.6. *The initial (pre-compilation) state of $\mathbb{J}\mathbb{K}$, call it z_{jk0} , satisfies the simple security condition, the $*$ -property, and the ds -property.*

Proof. Initially \mathbb{JK} contains only those **auth** statements from the individual policies \mathbb{J} and \mathbb{K} . By Theorem 6.1.5 \mathbb{J} and \mathbb{K} satisfy Definitions 2.2.2–2.2.4. Therefore the combined policy, with no additional authorizations is also secure. \square

Compiling the resulting code generates four discretionary authorizations and three system authorizations.

```

cando(KS1, J01, W);
cando(KS1, J02, W);
cando(KS2, J01, R);
cando(KS2, J02, R);
auth(KS2, J02, R);
auth(KS2, J02, W);
auth(KS2, J01, W);

```

The security of the composed system according to the definition of security in the component policies and the specification of security level relationships in the composition statements is established below.

A composed system is secure by definition when the system authorizations for component policies remain unchanged and any new authorizations follow the same restrictions as individual policies.

Theorem 6.1.7. *For WASL system \mathbb{JK} , only secure state changes occur during compilation.*

Proof. Let the state of \mathbb{J} prior to composition be $v_{\mathbb{J}} = (b_{\mathbb{J}}, f_{\mathbb{J}}, m_{\mathbb{J}})$ and of \mathbb{K} prior to composition be $v_{\mathbb{K}} = (b_{\mathbb{K}}, f_{\mathbb{K}}, m_{\mathbb{K}})$, where $v_{\mathbb{J}}$ and $v_{\mathbb{K}}$ are secure. After composition the states of these systems are $v'_{\mathbb{J}} = (b'_{\mathbb{J}}, f'_{\mathbb{J}}, m'_{\mathbb{J}})$ and $v'_{\mathbb{K}} = (b'_{\mathbb{K}}, f'_{\mathbb{K}}, m'_{\mathbb{K}})$, respectively.

Discretionary authorizations added to the composed policy include those generated by the rule `inlevel(s,11) & inlevel(o,12) & leveltype(11,K) & leveltype(12,J) => cando(s,o,a)` (each an entry in m). This rule states that new discretionary authorizations will have a subject from \mathbb{K} and an object from \mathbb{J} . This precludes the addition of entries to m that have a subject and object from the same system. Therefore, $m_{\mathbb{J}} = m'_{\mathbb{J}}$ with respect to subjects and objects in \mathbb{J} and $m_{\mathbb{K}} = m'_{\mathbb{K}}$ with respect to subjects and objects in \mathbb{K} .

\mathbb{J} and \mathbb{K} were previously compiled so $b_{\mathbb{J}} = b'_{\mathbb{J}}$ and $b_{\mathbb{K}} = b'_{\mathbb{K}}$ because a change in m is required before b can be changed. Additionally, $f_{\mathbb{J}}$ and $f_{\mathbb{K}}$ remain unchanged so $f_{\mathbb{J}} = f'_{\mathbb{J}}$ and $f_{\mathbb{K}} = f'_{\mathbb{K}}$. Therefore $v_{\mathbb{J}} = v'_{\mathbb{J}}$ and $v_{\mathbb{K}} = v'_{\mathbb{K}}$. $v_{\mathbb{J}}$ and $v_{\mathbb{K}}$ are known to be secure and, thus, $v'_{\mathbb{J}}$ and $v'_{\mathbb{K}}$ are also secure.

Definition 2.2.1 and the semantics of the relations `levelorder` and `levelgeq` establish $\mathbb{JS} \text{ dom } \mathbb{JU}$ and $\neg(\mathbb{JU} \text{ dom } \mathbb{JS})$.

The only way to add a system authorization (s, o, \underline{r}) for some s and o (i.e., $\text{auth}(s, o, \underline{R})$ in WASL) is for a discretionary authorization in m (a statement $\text{cando}(s, o, \underline{R})$) to be in compliance with the relation $f_S(s) \text{ dom } f_O(o)$ (equivalent to $\text{inlevel}(s, 11) \ \& \ \text{inlevel}(o, 12) \ \& \ \text{levelgeq}(11, 12)$). The WASL rule $\text{cando}(s, o, \underline{R}) \ \& \ \text{inlevel}(s, 11) \ \& \ \text{inlevel}(o, 12) \ \& \ \text{levelgeq}(11, 12) \Rightarrow \text{auth}(s, o, \underline{R})$ is the only way to add a system authorization for \underline{r} access and $\{s : s \in S\}$, thus meeting the requirements of Theorems 6.1.2–6.1.4.

The only way to add a system authorization (s, o, \underline{w}) for some s and o , ($\text{auth}(s, o, \underline{W})$ in WASL), where $s \in S$ and $o \in O$ is for a discretionary authorization in m (a statement $\text{cando}(s, o, \underline{W})$) to be in compliance with the relation $f_O(o) \text{ dom } f_S(s)$ (equivalent to $\text{inlevel}(s, 11) \ \& \ \text{inlevel}(o, 12) \ \& \ \text{levelgeq}(12, 11)$). The WASL rule $\text{cando}(s, o, \underline{W}) \ \& \ \text{inlevel}(s, 11) \ \& \ \text{inlevel}(o, 12) \ \& \ \text{levelgeq}(12, 11) \Rightarrow \text{auth}(s, o, \underline{W})$ is the only way to add a system authorization for \underline{w} access and $\{s : s \in S\} \subseteq S$, thus meeting the requirements of Theorems 6.1.2–6.1.4.

Other policy modifications (e.g., the addition of `inlevel` statements) do not change the security state of the system. Therefore, state transitions that occur during compilation are secure.

Thus, any additions to m are cross-system discretionary authorizations. New system authorizations can be made only in compliance with Definitions 2.2.2–2.2.4. Therefore, \mathbb{JK} remains secure by definition during compilation. \square

Could, however, information flow from \mathbb{J} to \mathbb{K} and then back to \mathbb{J} in a way that violates \mathbb{J} 's security policy? For example, suppose object `ObjectJ1` has classification level `JL1` and object `ObjectJ2` has classification level `JL2` where $\text{levelgeq}(\text{JL2}, \text{JL1})$ is true. Could some subject `SubjK1` read `ObjectJ2` and write the data to object `ObjectJ1`?

Theorem 6.1.7 addresses this possibility and ensures security is maintained. Policies being composed share a compatible definition of security and the *dom* relation apply to the authorizations generated during composition just as they do during compilation of the individual policies. If, however, security classifications are not properly assigned or the partial-ordering of classification levels is not maintained, these errors are identified during the compilation of the composition. Thus, no inappropriate data flows are possible and BLP-defined security is maintained.

6.2 *Biba's Model Formalized*

Biba's strict integrity policy [Bib77] maintains a notion of data integrity by answering the question, "Is the subject or data trustworthy enough for the specified activity?" The model (also called Biba's model or just Biba) does not allow lower integrity subjects to write information to higher integrity subjects. Similarly, high integrity subjects should not read lower integrity information.

This section considers a policy that conforms to these requirements, beginning with the formalisms in Section 6.2.1 and continuing with a demonstrated policy composition with associated proofs of security in Section 6.2.2.

6.2.1 Formalisms of Biba. The formalization of the model is presented in Section 6.2.1.1 with necessary theorems in Section 6.2.1.2.

6.2.1.1 Biba's Model: Formal Specification. This section uses the conventions, definitions, and descriptions above to formally define Biba's model following [Bib77] and [Bis03]. Properties identified define when a system is secure within this framework.

Axiom 6.2.1 specifies that a subject of the same or lower integrity level than an object may observe that object. This is the dual of the BLP *simple security condition*. This axiom prevents a subject from becoming corrupted by less trusted information and is identified in WASL by stating that an authorization for a subject (\mathbf{s}) to read an object (\mathbf{o}) requires the integrity level of the subject (11) to be dominated by the integrity level of the object (12): $\text{inlevel}(\mathbf{s},11) \ \& \ \text{inlevel}(\mathbf{o},12) \ \& \ \text{levelgeq}(12,11) \Rightarrow \text{auth}(\mathbf{s},\mathbf{o},\mathbf{R})$.

Axiom 6.2.1. $\forall s \in S, o \in O$ such that $(s, o, \mathbf{r}) \Rightarrow f_O(o) \text{ dom } f_S(s)$.

The second axiom states a subject of the same or relatively higher integrity level than an object may modify that object. This is the dual of the BLP **-property*; a subject may not corrupt an object trusted information. WASL code for this axiom identifies that an authorization for a subject (\mathbf{s}) to write to an object (\mathbf{o}) requires the integrity level of the subject (11) to dominate the integrity level of the object (12): $\text{inlevel}(\mathbf{s},11) \ \& \ \text{inlevel}(\mathbf{o},12) \ \& \ \text{levelgeq}(11,12) \Rightarrow \text{auth}(\mathbf{s},\mathbf{o},\mathbf{W})$.

Axiom 6.2.2. $\forall s \in S, o \in O$ such that $(s, o, \mathbf{w}) \Rightarrow f_S(s) \text{ dom } f_O(o)$.

The third axiom specifies that a subject, s_1 , may invoke another subject, s_2 , if s_1 has the same or a higher integrity level as s_2 . Data transfer from s_1 to s_2 is presumed and therefore a subject is prohibited from using a third party as a proxy to inject lower integrity information. The corresponding WASL code identifies that an authorization for one subject ($\mathbf{s1}$) to invoke another subject ($\mathbf{s2}$) requires the integrity level of the first (11) to dominate the integrity level of the second (12): $\text{inlevel}(\mathbf{s1},11) \ \& \ \text{inlevel}(\mathbf{s2},12) \ \& \ \text{levelgeq}(11,12) \Rightarrow \text{auth}(\mathbf{s1},\mathbf{s2},\mathbf{I})$.

Axiom 6.2.3. $\forall s_1, s_2 \in S$ such that $(s_1, s_2, \mathbf{i}) \Rightarrow f_S(s_1) \text{ dom } f_S(s_2)$.

The following definitions identify the secure states of a system and define state changes for Biba's model of security.

Definition 6.2.1. A system $\Sigma(b, z_0)$ is secure if and only if, in that system, every authorization $b_x \in b$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3.

Definition 6.2.2. A system $\Sigma(b, z_0)$ changes state any time an authorization $b_x \in b$ is added to or deleted from the set of system authorizations b .

When a system's state changes, the state is $\Sigma'(b', z_0)$ with the new set of authorizations b' . The axioms and definitions above are sufficient for making assertions about the integrity of a system according to Biba's model. The next section identifies the requirements for preserving integrity.

6.2.1.2 Security Theorems.

Theorem 6.2.1. Let $\Sigma(b, z_0)$ represent the a system where $b = \emptyset$. $\Sigma(b, z_0)$ preserves integrity.

Proof. There are no authorizations in the system. Therefore, Definition 6.2.1 holds trivially. \square

System authorizations b are captured in the `auth` statements of a WASL specification $\sigma(b, z_{\sigma 0})$ where $z_{\sigma 0}$ is the pre-compiled specification of $\sigma(b, z_{\sigma 0})$ that contains no `auth` statements.

Theorem 6.2.2. The initial (pre-compilation) state of a WASL system $\sigma(b, z_{\sigma 0})$ is secure.

Proof. State $z_{\sigma 0}$ initially contains no system authorizations—no `auth` statements—and, thus, conforms to Theorem 6.2.1 and satisfies Definition 6.2.1 trivially. \square

The next three theorems establish the security of a system when adding authorizations of each of the types permitted in Biba systems.

Theorem 6.2.3. Let b_1 be an authorization of the form (s, o, \underline{x}) and let $\Sigma b, z_0$ be the current, secure system where $b_1 \notin b$. $\Sigma'(b', z_0)$ is identical to $\Sigma(b, z_0)$ except the authorizations in $\Sigma'(b', z_0)$ are $b' = b \cup b_1$. System $\Sigma'(b, z_0)$ is secure if and only if $f_O(o) \text{ dom } f_S(s)$.

Proof. Assume $\neg(f_O(o) \text{ dom } f_S(s))$. Then $b_1 \in b'$ violates Axiom 6.2.1 and by Definition 6.2.1 the system $\Sigma'(b', z_0)$ is not secure. Thus it must be that $f_O(o) \text{ dom } f_S(s)$.

Assume $\Sigma'(b', z_0)$ is secure. Then every $b_x \in b'$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3. By Definition 6.2.1, every $b_x \in b$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3 and so each $b_x \in b'$ including $b_x = b_1$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3. This completes the proof. \square

Theorem 6.2.4. Let b_1 be an authorization of the form (s, o, \underline{w}) and let $\Sigma(b, z_0)$ be the current, secure system where $b_1 \notin b$. Furthermore, let the authorizations for $\Sigma(b, z_0)$ be b . $\Sigma'(b', z_0)$ is identical to $\Sigma(b, z_0)$ except the authorizations in Σ' are $b' = b \cup b_1$. System $\Sigma'(b', z_0)$ is secure if and only if $f_S(s) \text{ dom } f_O(o)$.

Proof. Assume $\neg(f_S(s) \text{ dom } f_O(o))$. Then $b_1 \in b'$ violates Axiom 6.2.2 and by Definition 6.2.1 the system $\Sigma'(b', z_0)$ is not secure. Thus it must be that $f_S(s) \text{ dom } f_O(o)$.

Assume $\Sigma'(b', z_0)$ is secure. Then every $b \in b'$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3. By Definition 6.2.1, every $b_x \in b$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3 and so each $b_x \in b'$ including $b_x = b_1$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3. This completes the proof. \square

Theorem 6.2.5. *Let b_1 be an authorization of the form (s_1, s_2, i) and let $\Sigma(b, z_0)$ be the current, secure system where $b_1 \notin b$. Furthermore, let the authorizations for $\Sigma(b, z_0)$ be b . Allow system $\Sigma'(b', z_0)$ to be identical to $\Sigma(b, z_0)$ with the exception that the authorizations in Σ' are $b' = b \cup b_1$. System $\Sigma'(b', z_0)$ is secure if and only if $f_S(s_1) \text{ dom } f_S(s_2)$.*

Proof. Assume $\neg(f_S(s_1) \text{ dom } f_S(s_2))$. Then $b_1 \in b'$ violates Axiom 6.2.2 and by Definition 6.2.1 the system $\Sigma'(b', z_0)$ is not secure. Thus it is necessary that $f_S(s_1) \text{ dom } f_S(s_2)$.

Assume $\Sigma'(b', z_0)$ is secure. Then every $b \in b'$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3. By Definition 6.2.1, every $b_x \in b$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3 and so each $b_x \in b'$ including $b_x = b_1$ satisfies Axioms 6.2.1, 6.2.2, and 6.2.3. This completes the proof. \square

Theorem 6.2.6 asserts that WASL constructs for a Biba-consistent system σ properly maintains system security during state changes that occur during compilation. A secure state change is one in which any added `auth` statements do not violate system security.

Theorem 6.2.6. *State changes that occur during compilation of initially-secure WASL system σ are secure.*

Proof. Definition 2.2.1 and the semantics of the relations `levelorder` and `levelgeq` establish that security levels are partially ordered.

In accordance with Axiom 6.2.1, the only way to securely add a system authorization, (s, o, r) for some s and o , (`auth(s,o,R)` in WASL), where $s \in S$ and $o \in O$ is for the integrity levels of s and o to be related as $f_O(o) \text{ dom } f_S(s)$ (equivalent to `inlevel(s,11) & inlevel(o,12) & levelgeq(12,11)`). The rule `inlevel(s,11) & inlevel(o,12) & levelgeq(12,11) => auth(s,o,R)` is the only way to add a system authorization for `r` access. Thus, Theorem 6.2.3 is satisfied.

In accordance with Axiom 6.2.2, the only way to securely add a system authorization, (s, o, w) for some s and o , (`auth(s,o,W)` in WASL), where $s \in S$ and $o \in O$ is for the integrity levels of s and o to be related as $f_S(s) \text{ dom } f_O(o)$ (equivalent to `inlevel(s,11) & inlevel(o,12) & levelgeq(11,12)`). The rule `inlevel(s,11) & inlevel(o,12) & levelgeq(11,12) => auth(s,o,W)` is the only way to add a system authorization for `w` access. Thus, Theorem 6.2.4 is satisfied.

In accordance with Axiom 6.2.3, the only way to securely add a system authorization, (s_1, s_2, i) for some s_1 and s_2 , (`auth(s1,s2,R)` in WASL), where $s_1 \in S$ and $s_2 \in S$ is for the integrity levels of s_1 and s_2 to be related as $f_S(s_1) \text{ dom } f_S(s_2)$ (equivalent to `inlevel(s1,11) & inlevel(12,12) &`

$\text{levelgeq}(l_1, l_2)$). The rule $\text{inlevel}(s_1, l_1) \ \& \ \text{inlevel}(s_2, l_2) \ \& \ \text{levelgeq}(l_1, l_2) \Rightarrow \text{auth}(s_1, s_2, w)$ is the only way to add a system authorization for \underline{i} access. Thus Theorem 6.2.5 applies.

Other policy modifications are made (e.g., the addition of `inlevel` statements), but they do not change the security state of the system. Therefore, state transitions that occur during compilation are secure. \square

Security defined for Biba as described above permits the accurate implementation of Biba-compliant policies in WASL. These policies can, in turn, have been proven to support Biba’s definitions as has been shown for any system σ . Three WASL rules, one generating system authorizations for each of *Observe*, *Modify*, and *Invoke*, are the enabling mechanism for this assurance of security. The above proofs can be extended without loss of generalization to any policy specification that uses these rules to generate system authorizations.

6.2.2 Composition of Biba Systems in WASL. Secure policy composition is also supported in WASL. Consider two individual policies, \mathbb{L} and \mathbb{M} . These policies are composed and the resulting policy \mathbb{LM} is shown to maintain security.

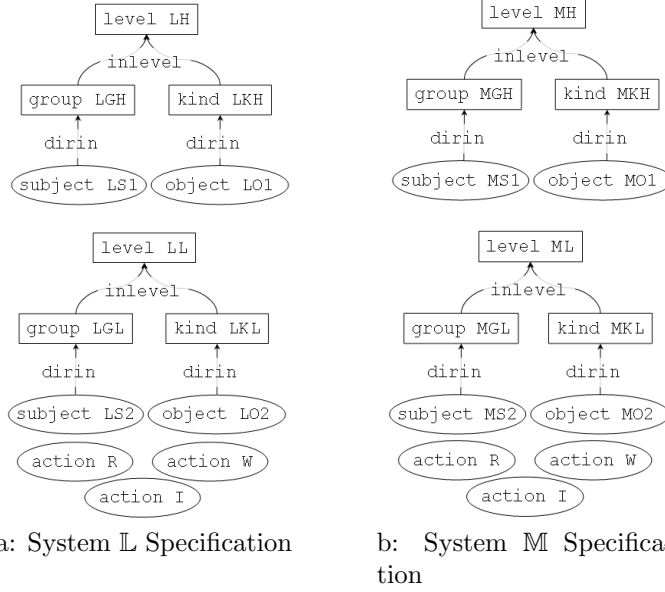
System \mathbb{L} has two integrity levels, two groups, one subject per group, two kinds, one object per kind, and the two actions \mathbb{R} (for *Observe*), \mathbb{W} (for *Modify*), and \mathbb{I} (for *Invoke*). Figure 20a depicts the specification of constants in the system and their relationships to each other. Constants that appear in ovals are used in system authorizations while those that do not are in boxes. Key terms identifying relations relating two constants are written across the lines that identify the relationship. The specification for system \mathbb{M} is similar, pictured in Figure 20b.

Policy specifications \mathbb{L} and \mathbb{M} include three WASL rules, one corresponding to each of the Axioms 6.2.1–6.2.3. Compilation of the programs yield system authorizations listed in Figure 20.

Policy composition presumes the two systems being composed share a compatible definition of security in the component policies (e.g., Biba). The composed policy must sustain the integrity of component policies by neither removing nor adding system authorizations for subjects with objects in either system. Added system authorizations must be for subjects in one system and objects in the other and the WASL rules generating system authorizations must be identical to those used by the systems that are to be composed (i.e., the three WASL–Biba rules).

Suppose the system administrators permit \mathbb{M} subjects access to \mathbb{L} objects and determine integrity level ML is equivalent to level LL . Generating the composed policy begins by combining the two previously-compiled policy specifications and removing any authorization-generating rules. This step is easily automated and the initial combination is secure as seen in Theorem 6.2.7.

Theorem 6.2.7. *The initial state of \mathbb{LM} , call it z_{jk_0} , is secure.*



```

auth(LS1, L01, R); auth(LS2, L01, R);
auth(LS1, L01, W); auth(LS2, L02, R);
auth(LS1, L02, W); auth(LS2, L02, W);

```

c: \mathbb{L} System Authorizations

```

auth(MS1, M01, R); auth(MS2, M01, R);
auth(MS1, M01, W); auth(MS2, M02, R);
auth(MS1, M02, W); auth(MS2, M02, W);

```

d: \mathbb{M} System Authorizations

Figure 20: Biba Systems \mathbb{L} and \mathbb{M} Specifications and Generated Authorizations

Proof. Initially \mathbb{LM} contains only those `auth` statements from the individual policies \mathbb{L} and \mathbb{M} . By definition and Theorems 6.2.2 and 6.2.6 \mathbb{L} and \mathbb{M} are secure. Therefore the policy resulting from combining policies \mathbb{L} and \mathbb{M} with no additional authorizations is also secure. \square

One rule is added to associate \mathbb{M} subjects at level ML with level LL .

```

-- relate LL to ML
inlevel(s,ML) => inlevel(s,LL);

```

Compiling the code resulting from the combination of \mathbb{L} with \mathbb{M} and the composition rule generates four new system authorizations.

```

auth(MS2, L01, R); auth(MS2, L02, R);
auth(MS2, L02, W); auth(MS2, L02, I);

```

A composed system is secure by definition when the system authorizations for component policies remain unchanged and any new authorizations follow the same restrictions as individual policies (following Axioms 6.2.1, 6.2.2, and 6.2.3).

Theorem 6.2.8. *For WASL system \mathbb{LM} , only secure state changes occur during compilation. Additions of `auth` statements comply with Axioms 6.2.1–6.2.3 and both \mathbb{L} and \mathbb{M} maintain their individual security.*

Proof. \mathbb{L} and \mathbb{M} are compiled prior to composition, generating all possible secure authorizations for subjects and objects within either system. Authorizations added to the policy during composition include only those generated for subjects newly-associated with integrity level \mathbb{L} (namely $\mathbb{MS2}$) from \mathbb{M} . Using the Biba-enforcing rules the only new authorizations generated during composition include objects in system \mathbb{L} —those with integrity levels associated with level \mathbb{L} . Thus, added authorizations are for subjects from \mathbb{M} and objects from \mathbb{L} , sustaining the security of individual policies \mathbb{L} and \mathbb{M} .

For authorizations added during composition, Theorems 6.2.3, 6.2.4, and 6.2.5 apply and security is shown to be preserved. Protection against indirect security violations where information passes securely, for example, from \mathbb{L} to \mathbb{M} but it is returned back into \mathbb{L} such that security is violated is treated in the discussion above with Theorem 6.2.8. \square

6.3 CW Model Formalized

The Chinese Wall (CW) security policy is a model that is neither a confidentiality (like BLP) nor an integrity (like Biba) security policy, but has properties of both. With respect to confidentiality, CW prevents unauthorized subjects from accessing data. On the other hand, any conflict of interest would breach integrity and CW prevents such breaches, which constitutes the integrity aspect of the system’s security. The formal model including proofs are in Section 6.3.1 with the WASL implementation following in Section 6.3.2.

6.3.1 Formalisms of CW. This section formalizes the CW model from [BN89]. It identifies in Section 6.3.1.1 the notation to be used and presents CW formalization in Section 6.3.1.2 with the corresponding WASL constructs.

6.3.1.1 The System: Notations. The symbols used for CW formalization and the WASL code specifying the corresponding construct is summarized in Table 18. Additional symbols include the conjunction and disjunction represented by \wedge and \vee , respectively. Variables used below are the positive integer indices $h, j, k, l \in \mathbb{N}$.

6.3.1.2 Chinese Wall: Formal Specification. This section defines CW following the formal model presentation in [BN89]. The properties herein identify when a system can be considered secure within this framework. The first definitions establish the access control matrix \underline{N} and the concept of sanitized information. The $\text{auth}(s, o, R)$ statements correspond to *true* entries in $\underline{N}(k, l)$ and serves as the control matrix in WASL. A sanitized data object o is specified in WASL by using the statement $\text{dirin}(o, \text{CDSan})$. A CW system is denoted as $\Sigma(\underline{N})$.

Table 18: Additional Symbols, Semantics, and WASL Encoding for CW

Sym	Description	WASL Representation
CD	The dataset type	<code>const kind CD</code>
COI	The conflict of interest class	<code>const kind COI</code>
L	Set of security labels, $\{(x, y)\}$	Set of <code>dirin(x,y)</code> when x is a kind, y is a kind, <code>dirin(x,COI)</code> , and <code>dirin(y,CD)</code>
\underline{N}	$\{\ S\ \times \ O\ \}$ matrix with boolean entries for each (h, j) ($s_h \in S$ and $o_j \in O$) identifying whether or not s_h has access to o_j ; $h, j \in \mathbb{N}$	Set of <code>auth(u,o,R)</code> statements
$X(o)$	Function returning x for object o	Identified by <code>in(o,K)</code> when K is declared as <code>const kind K</code> and <code>dirin(K,COI)</code>
x	A COI class	<code>const kind K</code> where <code>dirin(K,COI)</code>
x_o	The sanitized COI	<code>const kind COISan</code>
(x, y)	A security label, $(x, y) \in L$	Pair <code>dirin(Ky,Kx)</code> when <code>dirin(Ky,CD)</code> and <code>dirin(Kx,COI)</code>
$Y(o)$	Function returning y for o	Identified by <code>in(O,K)</code> when K is declared as <code>const kind K</code> and <code>dirin(K,CD)</code>
y	A dataset	<code>dirin(y,CD)</code>
y_o	The sanitized CD	<code>const kind CDSan</code>

As is mentioned above, WSN nodes operate autonomously with only periodic control exerted by the gateway computer. The limitations on communications and memory and computational capacities means there cannot be a central arbiter determining responses to access requests. Nor is there sufficient capacity at the nodes to maintain a data store of all subject and object relationships, the rules for their interaction, or to compute proper response for each request.

Therefore, if no CD in a particular COI is included in the initial authorizations for a given subject, no CD in that COI is accessible to that subject. Axiom 6.3.1 replaces Axiom 2.2.3 to reflect this more-restrictive paradigm. WASL identifies initial permissions using the `do` authorization statement construct and then applies the rule `dirin(o1,CDSan) | (do(s,o2,R) & in(o1,k1) & in(o2,k1) & dirin(k1,CD)) => auth(s,o1,R)` to enforce Axiom 6.3.1.

Axiom 6.3.1. $r(k, l) \rightarrow \underline{y} \iff \forall \underline{N}(k, j) = true, (y_j = y_l)$

The initial state of a WASL system σ that has implemented the Chinese Wall is established by proving that it complies with Axiom 2.2.4.

Theorem 6.3.1. *The initial state of some WASL system σ , call it $z_{\sigma 0}$, is secure.*

Proof. \underline{N} is represented by the set of `auth` statements. System σ has no `auth` statements prior to compilation. Thus, by Axiom 2.2.4 σ is secure. \square

The following axiom defines when a *write* (or *modify* in Biba) is permitted. Two rules in WASL enforce this requirement. The first rule identifies the unauthorized *writes*: $\text{-in}(\text{o2, San}) \ \& \ ((\text{in}(\text{o1, k1}) \ \& \ \text{in}(\text{o2, k2}) \ \& \ \text{dirin}(\text{k1, CD}) \ \& \ \text{dirin}(\text{k2, CD}) \ \& \ \text{-equals}(\text{k1, k2})) \ \& \ \text{do}(\text{s, o2, R})) \Rightarrow \text{do}(\text{s, o1, -W})$. Using this rule and the authorizations to *read* as identified above, the second rule generates the *write* authorizations for the system: $\text{-do}(\text{s, o1, -W}) \ \& \ \text{auth}(\text{s, o1, R}) \Rightarrow \text{auth}(\text{s, o1, W})$.

Axiom 6.3.2. Write access to any object o_j by any subject s_k is permitted if and only if $\underline{N}'(k, j) = \text{true}$ and there does not exist any object o_h for which $(\underline{N}'(k, h) = \text{true})$ when it can be read by s_k where $y_h \neq y_j \wedge y_h \neq y_o$.

Theorem 6.3.2 asserts that WASL constructs and composition properly maintain system security. Secure state changes are those that comply with Axioms 2.2.3 and 6.3.2.

Theorem 6.3.2. *State changes that occur during compilation of some initially-secure WASL system σ are secure.*

Proof. Read authorizations in σ :

One WASL rule is the only way to add a *read* authorization to σ . This rule thus provides both the necessary and sufficient condition for adding *read* authorizations.

Axiom 6.3.1 identifies the necessary and sufficient condition for adding a *read* authorization. The WASL expression $\text{dirin}(\text{o1, CDSan}) \ | \ (\text{do}(\text{s, o2, R}) \ \& \ \text{in}(\text{o1, k1}) \ \& \ \text{in}(\text{o2, k1}) \ \& \ \text{dirin}(\text{k1, CD}))$ identifies all the objects o1 that are sanitized (and by definition accessible by all subjects) and that are not in the dataset of a permitted *read*. Any match between a subject s and object o1 in this expression results in the addition of a system authorization $\text{auth}(\text{s, o1, R})$ to the policy. The conditions of the WASL *read* rule are, therefore, equivalent to the requirements of Axiom 6.3.1.

Write authorizations in \mathbb{P} :

The first condition in Axiom 6.3.2 that permits a *write* authorization is $\underline{N}'(k, j) = \text{true}$. The WASL rule permitting subject s to write object o includes the imperative that $\text{auth}(\text{s, o, R})$ be specified in the policy, meeting this condition.

The converse of the second condition of Axiom 6.3.2 specifies that no subject may *write* to an object if the subject has an authorization to *read* any objects in a different, non-sanitized CD. These disallowed *writes* match those in the expression $\text{-in}(\text{o2, CDSan}) \ \& \ ((\text{in}(\text{o, k1}) \ \& \ \text{in}(\text{o2, k2}) \ \& \ \text{dirin}(\text{k1, CD}) \ \& \ \text{dirin}(\text{k2, CD}) \ \& \ \text{-equals}(\text{k1, k2})) \ \& \ \text{auth}(\text{s, o2, R}))$, a condition that yields the required negative authorization $\text{do}(\text{u, o, -W})$; for each u and o1 to which it applies. Having identified the cases when a *write* must not be permitted, the absence of this indicator, $\text{-do}(\text{s, o, -W})$, is the

required condition. This logic is included in the WASL *write* rule as the second mandatory condition of Axiom 6.3.2.

Both requirements of Axiom 6.3.2 are necessary for a *write* authorization to be added to \mathbb{P} and are established prior to the addition of and compilation of the WASL *write* rule, thus preserving security. In addition, this WASL rule is sufficient for adding *write* authorizations to the system as it is the only means of adding $\text{auth}(s, o, w)$ statements to \mathbb{P} . \square

6.3.2 WASL Implementation. Policy \mathbb{P} is an implementation of a policy enforcing the principles of the CW model. WASL entities are related to CW as summarized in Table 18. Subjects, objects, actions, and kinds (reflecting COIs and CDs) and the relationships among them all have direct correlation with CW model formalisms.

System \mathbb{P} has two COIs, *COI1* and *COI2*, with two datasets in each, *CD1* and *CD2* in *COI1* and *CD3* and *CD4* in *COI2*. Sanitized objects are maintained in a separate COI and dataset, *COISan* and *CDSan*, respectively. Figure 21 identifies the WASL CD/COI hierarchy while Figure 22 identifies objects and their CDs for modeling this system. Relevant code to depicted elements is provided in the figures but the complete code corresponding to these figures follows.

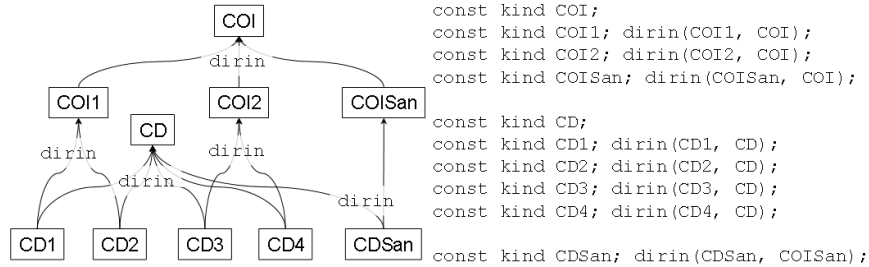


Figure 21: CD/COI Structure for System \mathbb{P}

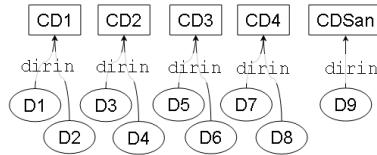


Figure 22: Object/CD Structure for System \mathbb{P}

```

begin
-- declare fundamental COI and CD kinds
const kind COI; const kind CD;
-- declare COIs and associate them with COI
const kind COI1; const kind COI2;
dirin(COI1, COI); dirin(COI2, COI);
const kind San; dirin(San, COI);

```

```

-- declare CDs and associate them with CD
const kind CD1; const kind CD2;
const kind CD3; const kind CD4;
dirin(CD1, CD); dirin(CD2, CD);
dirin(CD3, CD); dirin(CD4, CD);
dirin(San, CD);
--associate CDs with COIs
dirin(CD1, COI1); dirin(CD2, COI1);
dirin(CD3, COI2); dirin(CD4, COI2);
-- declare objects and associate them with CDs
const object D1; const object D2;
const object D3; const object D4;
const object D5; const object D6;
const object D7; const object D8;
dirin(D1, CD1); dirin(D2, CD1);
dirin(D3, CD2); dirin(D4, CD2);
dirin(D5, CD3); dirin(D6, CD3);
dirin(D7, CD4); dirin(D8, CD4);
const object D9; dirin(D9, San);
end;

```

There are eight subjects, each with some initially-specified access to different combinations of objects as shown in Figure 23. Accesses (all *read* at this point) are defined in WASL using the `do(s,o,R)` construct. The code expressing these and the added subjects follows. This code includes a declaration of the two actions in the system, *Read* and *Write*.

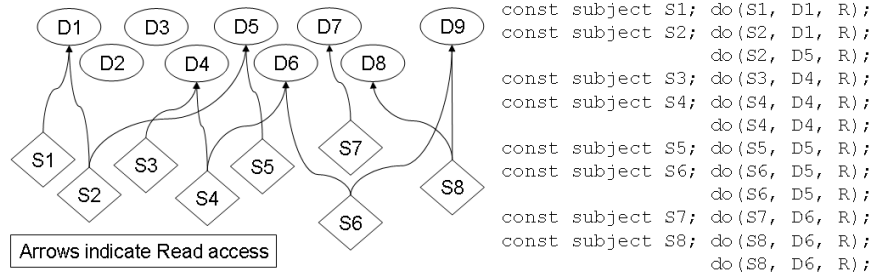


Figure 23: Initial Subject Accesses in System \mathbb{P}

```

-- declare subjects
const subject S1; const subject S2;
const subject S3; const subject S4;
const subject S5; const subject S6;
const subject S7; const subject S8;
-- declare actions
const action R; const action W;
-- establish initial accesses
do(S1, D1, R); do(S2, D1, R);
do(S2, D5, R); do(S3, D4, R);
do(S4, D4, R); do(S4, D6, R);
do(S5, D5, R); do(S6, D6, R);
do(S6, D9, R); do(S7, D7, R);
do(S8, D8, R); do(S8, D9, R);

```


Compiling the above code with the three WASL rules leads to the following system authorizations, permissions that satisfy Theorems 6.3.1 and 6.3.2. Thus, \mathbb{P} is secure by CW definitions.

```

auth(S1, D1, R); auth(S1, D2, R); auth(S1, D9, R); auth(S1, D1, W); auth(S1, D2, W);
auth(S2, D1, R); auth(S2, D2, R); auth(S2, D5, R); auth(S2, D6, R); auth(S2, D9, R);
auth(S3, D3, R); auth(S3, D4, R); auth(S3, D9, R); auth(S3, D3, W); auth(S3, D4, W);
auth(S4, D3, R); auth(S4, D4, R); auth(S4, D5, R); auth(S4, D6, R); auth(S4, D9, R);
auth(S5, D5, R); auth(S5, D6, R); auth(S5, D9, R); auth(S5, D5, W); auth(S5, D6, W);
auth(S6, D5, R); auth(S6, D6, R); auth(S6, D9, R); auth(S6, D5, W); auth(S6, D6, W);
auth(S7, D7, R); auth(S7, D8, R); auth(S7, D9, R); auth(S7, D7, W); auth(S7, D8, W);
auth(S8, D7, R); auth(S8, D8, R); auth(S8, D9, R); auth(S8, D7, W); auth(S8, D8, W);

```

Two WASL rules, one generating system authorizations for each of *read* and *write*, are the enabling mechanisms for the assurance of security. Policy composition for a CW system could be explored here, but the similarities with that shown in Sections 6.1.1 and 6.2.1 render such a presentation redundant.

6.4 Conclusion

The key to security enforcement in WASL is the construction of WASL rules and the proper selection of WASL constructs by the system administrator to express intended accesses for system subjects. Formal models of BLP, Biba, and CW provide the backdrop for rigorous analysis of WASL constructs. For these axiomatic systems, WASL is shown to enforce the requirements for sample policies and, without loss of generality, any policies following the rules and procedures described.

Policy composition includes the combination of two security policies that share a definition of security. With the addition of capabilities for subjects in one policy to access objects in the other, the composition is shown to sustain security for the component policies while assuring security for the composition as defined.

VII. Conclusion

This research has shown that a WSN operates more securely when a policy management system regulates accesses of subjects to particular data objects. The security system is even more powerful due to additional flexibility when the policies are managed and distributed independently from the mechanisms enforcing those policies, authentication, encryption, and other security measures.

WASL is the first policy language to provide a means for implementing just such a system for WSNs. The merits of the language are identified in Section 7.1 with recommendations for further study following in Section 7.2

7.1 Contributions

The core of this research is WASL, a language designed to represent security policies for WSNs. It is fully specified including the semantics of the various constructs that make use of a complete grammar. This section reviews aspects identified in the statement of purpose (Section 1.3) to identify the achievement of the stated objectives.

WASL demonstrates three well-studied and fully defined policies as well as hybrid policies combining the characteristics of those three. Additionally, any selected set of system authorizations or rules generating system authorizations could be designated to define a desired security policy, thus demonstrating the versatility of the language.

While not the primary objective of this work, the compression of policy specifications such that transmission to and storage at a node was shown but is an area addressed further below. The minimum information required at a node is the set of system authorizations, information easily extractable from a fully specified security policy.

The compiler for WASL uses statements and rules to generate the complete delineation of system authorizations as required. The concepts of discretionary and mandatory access control appear in Biba and BLP policy models, and the WASL construct is shown to be adaptable to the Chinese Wall model. Role-based authorizations have not been the focus of this work, but constructs are supported as they parallel those already discussed.

The language capably represents relationships among subjects, among users, and between these and security/classification levels. Groupings referenced in authorization statements imply the given authorization applies to every element in the group, simplifying the security administrator's policy specification task. Similarly, an association between a security level and an entity grouping applies the same relationship between that level and any element of the group. By demonstrating the hybridization of policies, the simultaneous enforcement of multiple partial orders of security levels

is exemplified. This flexibility adds capability to composition requirements, enabling a security administrator to superimpose a policy from another network onto the policy of the WSN.

Policy composition is secure when component policies remain unchanged directly and indirectly with respect to data flow. WASL supports secure composition of a variety of policies whether those policies have similar or dissimilar security models. A methodology for merging policies whose rules do not match is developed and demonstrated, with the security of the systems shown to be maintained in such circumstances.

Because the evaluated policy models are axiomatic systems, proofs of security are straightforward. These proofs are provided and are necessary for the assurance of security. The proofs apply equally to policy models in theory, to initial WASL-coded policy specifications, to the compiled policies, and to compositions of independent policies. They provide the basis for the claim that WASL policy compilations and compositions maintain security.

7.2 Recommendations for Future Work

This research advocates the distribution of system authorizations, and only system authorizations, throughout the WSN to manage periodically changing access privileges. While the system has been demonstrated and its feasibility substantiated, some aspects of its implementation would benefit from further investigation.

The compiler manages policy compilation and composition, but the compilation run times at a gateway can become prohibitively long using this JavaTM-based compiler when a large number of constant substitutions are required. Compiler efficiencies may be identified by using additional heuristics or through implementation using a declarative language paradigm.

Policies containing a large number of system authorizations can be represented in a sufficiently compact manner for distribution and storage among and within WSN motes. Efficiency is of the highest import, however, and the manifest representation can undoubtedly be improved.

Similarly, a table-lookup system at each node to assess a query's merit is assumed. The particular mechanism used to accomplish this task has not been addressed. This development should be performed in conjunction with data compression improvements as it is likely that the latter will be coupled with the former just as the gateway is coupled with the motes.

Finally, query-routing is a significant issue that must be investigated. The proposed policy-enforcement approach to security identifies how a node may respond when presented with a query it is qualified to answer. This research did identify possible responses of a node when it is not the intended recipient of the query, but did not explore the implications of the various courses of action.

Bibliography

- AIL05. Madhukar Anand, Zachary Ives, and Insup Lee. Quantifying eavesdropping vulnerability in sensor networks. In *Proceedings of the 2nd International Workshop on Data Management for Sensor Networks*, pages 3–9, New York NY, USA, 2005. ACM Press.
- AL93. Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.
- AMN02. Xuhui Ao, Naftaly H. Minsky, and Thu D. Nguyen. A hierarchical policy specification language and enforcement mechanism for governing digital enterprises. In *POLICY: 3rd International Workshop on Policies for Distributed Systems and Networks*, pages 38–49. IEEE Computer Society, Jun 2002.
- App98. Andrew W. Appel. *Modern Compiler Implementation in Java*. Cambridge University Press, New York NY, 1998.
- AUJP04. Sasikanth Avancha, Jeffrey Undercoffer, Anupam Joshi, and John Pinkston. Security for wireless sensor networks. In C. S. Raghavendra, Krishna M. Sivalingam, and Taieb Znati, editors, *Wireless Sensor Networks*, chapter 12. Kluwer Academic Publishers, Boston MA, 2004.
- BdVS02. Piero Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An algebra for composing access control policies. *ACM Transactions on Information System Security*, 5(1):1–35, 2002.
- BE02. David Braginsky and Deborah Estrin. Rumor routing algorithm for sensor networks. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 22–31, New York NY, USA, 2002. ACM Press.
- BGS92. John A. Bull, Li Gong, and Karen R. Sollins. Towards security in an open systems federation. In *Proceedings of the Second European Symposium on Research in Computer Security*, pages 3–20, London, UK, 1992. Springer-Verlag.
- BGS00. P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communications*, 7(5):10–15, Oct 2000.
- Bib77. K. J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corp., Bedford MA, April 1977.
- Bis03. Matt Bishop. *Computer Security—Art and Science*. Addison-Wesley, November 2003.
- BL75. D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE Corp., Bedford MA, July 1975.

- BN89. David F.C. Brewer and Michael J. Nash. The chinese wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214, May 1989.
- BY94. William R. Bevier and William D. Young. A state-based approach to noninterference. In *Proceedings of the Computer Security Foundations Workshop VII, CSFW 7*, pages 11–21, June 1994. <http://ieeexplore.ieee.org/iel2/986/7619/00315951.pdf>.
- CC97. Laurence Cholvy and Frédéric Cuppens. Analyzing consistency of security policies. In *18th IEEE Computer Society Symposium on Research in Security and Privacy*, 1997.
- CKP03. Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 551–562, New York NY, USA, 2003. ACM Press.
- CP03. Haowen Chan and Adrian Perrig. *Computer*, chapter Security and Privacy in Sensor Networks, pages 103–105. IEEE Press, October 2003.
- Cro07. Crossbow. *MICA2 Datasheet*. Crossbow Technology Incorporated, 4145 N. First Street, San Jose CA 95134, 2007. www.xbow.com.
- DDH⁺05. Wenliang Du, Jing Deng, Yungshiang S. Han, Pramod K. Varshney, Jonathan Katz, and Aram Khalili. A pairwise key predistribution scheme for wireless sensor networks. *ACM Transactions of Information Systems Security*, 8(2):228–258, 2005.
- DDLS01. Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Lecture Notes in Computer Science*, pages 18–39. Springer-Verlag, January 2001.
- DeT02. J. DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 105–113, 2002. <http://ieeexplore.ieee.org/iel5/7873/21681/01004365.pdf>.
- DSB04. Chairman DSB. Defense science board study on unmanned aerial vehicles and uninhabited combat air vehicles. Technical report, Office of the Undersecretary of Defense for Acquisition, Technology, and Logistics, February 2004.
- EN03. Eiman Elnahrawy and Badri Nath. Cleaning and querying noisy sensors. In *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 78–87, New York NY, USA, 2003. ACM Press.
- FG95. Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–34, 1994/1995.

- FG01. Riccardo Focardi and Roberto Gorrieri, editors. *Foundations of Security Analysis and Design: Tutorial Lectures*, volume 2171 of *Lecture Notes in Computer Science*, chapter Classification of Security Properties (Part I: Information Flow), pages 331–396. Springer, Berlin / Heidelberg, 2001.
- FSCY96. H. Feinstein, R. Sandhu, E. Coyne, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
- GG99. Virgil D. Gligor and Serban I. Gavrilă. Application-oriented security policies and their composition. *Lecture Notes in Computer Science*, 1550:67–74, 1999.
- GGF98. Virgil D. Gligor, Serban I. Gavrilă, and David Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *19th IEEE Computer Society Symposium on Research in Security and Privacy*, 1998.
- GKFS01. S. Gavrilă, D. R. Kuhn, D. F. Ferraiolo, and R. Sandhu. Proposed nist standard for role-based access control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, 2001.
- GLCB03. David Gay, Philip Levis, David Culler, and Eric Brewer. *The nesC Language: A Holistic Approach to Networked Embedded Systems*. University of California, Berkeley, and Intel Research, Berkeley CA, May 2003.
- GLvB⁺03. David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems, 2003.
- GM82. J. A. Gougen and J. Meseguer. Security policies and security models. In *Proceedings of the 1982 Symposium on Privacy and Security*, pages 11–20, 1982.
- GQ94. Li Gong and Xiaolei Qian. The complexity and composability of secure interoperation. In *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, pages 190–200, 1994.
- GR97. Rosario Gennaro and Pankaj Rohatgi. How to sign digital streams. In Burt Kaliski, editor, *Advances in Cryptology — Crypto '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 180–197. Springer-Verlag, Berlin, 1997.
- GS04. Saurabh Ganeriwal and Mani B. Srivastava. Reputation-based framework for high integrity sensor networks. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Networks*, pages 66–77, New York NY, USA, 2004. ACM Press.
- HSW⁺00. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth Inter-*

national Conference on Architectural Support for Programming Languages and Operating Systems, pages 93–104, New York NY, USA, 2000. ACM Press.

- JM92. Q. Shi J.A. McDerimid. Secure composition of systems. In *Proceedings of the Eighth Annual Computer Security Applications Conference*, pages 112–122, December 1992.
- JSS97. Sushil Jajodia, Pierangela Samarati, and V.S. Subrahmanian. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 31–42. IEEE, May 1997. INSPEC Accession Number: 5602909.
- KHJ02. Håkan Kvarnström, Hans Hedbom, and Erland Jonsson. A protection scheme for security policies in ubiquitous environments using one–way functions, August 2002. <http://www.teco.edu/~philip/ubicomp2002ws/organize/chalmers.pdf>.
- KSP06. Dong Seong Kim, Khaja Mohammad Shazzad, and Jong Sou Park. A framework of survivability model for wireless sensor network. In *Proceedings of the First International Conference on Availability, Reliability and Security*. IEEE, September 2006.
- KSW04. Chris Karlof, Naveen Sastry, and David Wagner. *TinySec: User Manual*. University of California, Berkeley, June 2004.
- Lam79. Leslie Lamport. Constructing digital signatures from a one–way function. Technical report, SRI International, October 1979.
- LB73. Leonard J. LaPadula and D. Elliott Bell. Secure computer systems: Mathematical foundations volume ii. Technical Report MTR-2547, MITRE Corp., Bedford MA, May 1973.
- LL03. Philip Levis and Nelson Lee. *TOSSIM: A Simulator for TinyOS Networks*. University of California, Berkeley, 1.0 edition, September 2003.
- LMG⁺04. Philip Levis, Sam Madden, David Gray, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in tinyos. In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation*, pages 1–14, 2004.
- McC87. Daryl McCullough. Specifications for multi–level security and a hook–up property. *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 161–166, 1987.
- McC88. Daryl McCullough. Noninterference and the composability of security properties. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 177–186, April 1988.

- McC90. Daryl McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, June 1990.
- McL92. John McLean. Proving noninterference and functional correctness using traces. *Journal of Computer Security*, 1(1), 1992.
- Mil94. Jonathan K. Millen. Unwinding forward correctability. *Proceedings of the Computer Security Foundations Workshop*, 3(1):2–10, 1994.
- NK93. Michael J. Nash and Ronald J. Kennett. Security policy in a complex logistics procurement. In *Proceedings of the Ninth Annual Computer Security Applications Conference*, pages 46–53, December 1993.
- PCST01. Adrian Perrig, Ran Canetti, Dawn Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Proceedings of the Internet Society Network and Distributed System Security Symposium*, pages 35–46, February 2001.
- PSSC04. Joseph Polastre, Robert Szewczyk, Cory Sharp, and David Culler. The mote revolution: Low power wireless sensor network devices. Symposium on High Performance Chips, IEEE Technical Committee on Microprocessors and Microcomputers, August 2004.
- PSW⁺01. Adrian Perrig, Robert Szewczyk, Victor Wen, David E. Culler, and J. D. Tygar. SPINS: security protocols for sensor networks. In *Mobile Computing and Networking*, pages 189–199, 2001.
- PSW04. Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, 2004.
- RN02. Tatyana Ryutov and B. Clifford Neuman. The specification and enforcement of advanced security policies. In *POLICY: 3rd International Workshop on Policies for Distributed Systems and Networks*, pages 128–138. IEEE Computer Society, Jun 2002.
- Roh99. Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.
- RSZ04. C. S. Raghavendra, Krishna M. Sivalingam, and Taieb Znati, editors. *Wireless Sensor Networks*. Kluwer Academic Publishers, Boston MA, 2004.
- RZFG99. Carlos Ribeiro, André Zúquete, Paulo Ferreira, and Paulo Guedes. Spl: An access control language for security policies with complex constraints. Technical Report RT/0001/99, INESC, Jan 1999.

- SCFY96. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, February 1996.
- SOBAC04. Weilian Su, Özgür B. Akan, and Erdal Cayirci. Communication protocols for sensor networks. In C. S. Raghavendra, Krishna M. Sivalingam, and Taieb Znati, editors, *Wireless Sensor Networks*, chapter 2. Kluwer Academic Publishers, Boston MA, 2004.
- Sta04. John A. Stankovic. Research challenges for wireless sensor networks. *ACM SIGBED Review*, 1(2):9–12, Jul 2004.
- Sut86. David Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, September 1986.
- UBJ⁺04. Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Matthew Johnson, Austin Tate, Jeff Dalton, and Stuart Aitken. Policy and contract management for semantic web services. In Terry Payne, editor, *Semantic Web Services: Papers from the 2004 Spring Symposium*, pages 121–128. AAAI, Menlo Park CA, USA, 2004.
- Zak96. Aris Zakynthinos. *On The Composition Of Security Properties*. PhD thesis, University of Toronto, 1996.
- ZG04. Feng Zhao and Leonidas J. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann Publishers, San Francisco CA, 2004.
- ZP04. G. Zhang and M. Parashar. Context-aware dynamic access control for pervasive computing, 2004.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY) 27-03-2008		2. REPORT TYPE PhD Dissertation		3. DATES COVERED (From — To) Sep 04-Mar 08	
4. TITLE AND SUBTITLE Composable Distributed Access Control and Integrity Policies for Query-Based Wireless Sensor Networks				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Marsh, David W.				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management 2950 Hobson Way WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/DEE/ENG/08-06	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Kenneth Littlejohn/Michelle Cheatham Air Force Research Laboratory (RYTC/RYSB) 2241 Avionics Circle Wright-Patterson AFB, OH 45433-7132 937-255-4947 X3587/x3162 Kenneth.Littlejohn/Michelle.Cheatham@wpafb.af.mil				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT An expected requirement of wireless sensor networks (WSN) is the support of a vast number of users while permitting limited access privileges. While WSN nodes have severe resource constraints, WSNs will need to restrict access to data, enforcing security policies to protect data within WSNs. To date, WSN security has largely been based on encryption and authentication schemes. WSN Authorization Specification Language (WASL) is specified and implemented using tools coded in Java™. WASL is a mechanism-independent policy language that can specify arbitrary, composable security policies. The construction, hybridization, and composition of well-known security models is demonstrated and shown to preserve security while providing for modifications to permit inter-network accesses with no more impact on the WSN nodes than any other policy update. Using WASL and a naïve data compression scheme, a multi-level security policy for a 1000-node network requires 66 bytes of memory per node. This can reasonably be distributed throughout a WSN. The compilation of a variety of policy compositions are shown to be feasible using a notebook-class computer like that expected to be performing typical WSN management responsibilities.					
15. SUBJECT TERMS Computer Security; Computer Access Control; Network Security; Information Security					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Rusty Baldwin, ENG, rbaldwin@afit.edu
U	U	U	UU	113	19b. TELEPHONE NUMBER (include area code) (937) 255-6565 ext. 4445