MODELS FOR PRIVACY IN UBIQUITOUS COMPUTING ENVIRONMENTS

BY

APU CHANDRASEN KAPADIA

B.S., University of Illinois at Urbana-Champaign, 1998
M.S., University of Illinois at Urbana-Champaign, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

This thesis addresses the discretionary privacy demands of users in heterogeneous distributed systems such as ubiquitous computing environments. Because of the physical proximity and pervasiveness of personal devices, sensors, actuators, and other devices and services, ubiquitous computing environments need a powerful infrastructure for coordinating accesses to these resources. However, this infrastructure makes it easy for malicious administrators to gain access to private information of users. We present models for privacy of a user's communication, unlinkability of a user's accesses, and authorized policy feedback that is both useful and privacy preserving. Our models expose the potential threats to a user's privacy, and allow users to express their individual and differing privacy demands based on these threats. We show how a user's privacy policies can be efficiently satisfied under our models.

For secure and private communication, we present a model for trustworthy routing, with a policy specification language that is computationally efficient to enforce. We show how quantitative trust models can be used to find trustworthy paths of communication and explore various semantic models of trust. For the unlinkability of a user's accesses to services in a ubiquitous computing environment, we present a model based on access control and decentralized enforcement of policy constraints. We prove that our solution is secure, and show how security can be maintained by trading off precision for evolving protection state. Lastly, we present a model called *Know* for providing feedback regarding access control decisions to users. This model aims to make ubiquitous computing environments more usable and secure, while honoring the privacy of other users in the system. Administrators can specify meta-policies to tailor feedback to individual users based on perceived threat to the policy's contents.

To my parents
Who valued my education
Above all else

Acknowledgments

I would like to thank the many people who have helped me during my career as a "professional student."

My advisor, Roy H. Campbell, for taking me under his wing while I was a young undergraduate student and stimulating my interest in research. For countless hours of memorable and thought-provoking discussions, and for his never-ending flow of ideas. And lastly, for always giving me the freedom to explore new and exciting research directions of my choosing.

The Department of Energy and the people at Krell for granting me the four year High-Performance Computer Science Fellowship, supported by Los Alamos National Laboratory, Lawrence Livermore National Laboratory, and Sandia National Laboratories. This fellowship funded my work on several interesting research projects and I am truly grateful for their support.

My committee members. Mahesh Viswanathan, who helped me refine and enhance my proposed models. Marianne Winslett and William H. Sanders for their insightful suggestions and feedback.

Anda Ohlsson for her invaluable help with administrative issues.

Chandra Chekuri, John Fischer, Sariel Har-Peled, Viraj Kumar, Kevin Milans, Shripad Thite, and Erin Wolf for helping me with several theoretical aspects of my thesis. Eyal Amir, Jodie Boyer, Susan Hinrichs, Kiran Lakkaraju, Adam Lee, Suvda Myagmar, Vasin Punyakanok, Anand Ranganathan, Nick Rizzolo, Cigdem Sengul, Samarth Swarup, and Dav Zimak for their helpful suggestions. Vijay Gupta, for his early advice on classes and graduate school. Wu-chun Feng, for mentoring me over two summer internships at Los Alamos National Laboratory, and for introducing me to networking research and the mountains of New Mexico and Colorado.

Jalal Al-Muhtadi, Prasad Naldurg, Geetanjali Sampemane, and Seung Yi, my dear friends and colleagues at the Systems Software Research Group for endless hours of animated and enriching discussions.

My wife, Phoebe Wolfskill, for her support and encouragement through all the phases of graduate school, for pampering me during the final months leading up to my defense, and for her immense love that is far beyond what I had ever hoped for.

My father, Sen Kapadia, for emphasizing values and aesthetics that have shaped my personality, for his enthusiasm for higher education that has carried me this far, and for his sense of humor that has taught me to enjoy life without a "sense of tumor."

My mother, Asha Kapadia, for her deep involvement in my education, for her unconditional and abundant love, for her selfless generosity to me and others, and for her undying support for anything I have ever wished to achieve or accomplish.

Table of Contents

List of Figures

List of Abbreviations

API	Application Programming Interface
BGP	Border Gateway Protocol
CA	Certificate Authority
CLTL	Constraint Linear Temporal Logic
COI	Conflict of Interest
CORBA	Common Object Request Broker Architecture
DAC	Discretionary Access Control
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
DDoS	Distributed Denial of Service
FTA	Fault Tree Analysis
ID	Identity
IDS	Intrusion Detection System
IOI	Item of Interest
IW	Inference Web
LTL	Linear Temporal Logic
MAC	Mandatory Access Control
MLS	Multi Level Security
MPLS	Multiprotocol Label Switching
NAT	Network Address Translator
OBDD	Ordered Binary Decision Diagram
PBN	Policy Based Network/Networking
PDA	Personal Digital Assistant
PDP	Policy Decision Point
PEP	Policy Enforcement Point

PKI	Public Key Infrastructure
PNS	Policy Negotiation Server
PRA	Permission Role Assignment
QoP	Quality of Protection
QoS	Quality of Service
RBAC	Role-Based Access Control
RFID	Radio Frequency Identification
SCC	Strongly Connected Component
SoD	Separation of Duty
URA	User Role Assignment
VPN	Virtual Private Network

1 Introduction

This thesis addresses the discretionary security and privacy demands of users in heterogeneous distributed systems such as *ubiquitous computing environments*. The goal of ubiquitous computing is to blend the computational or virtual environment into the everyday physical environment and to provide a mechanism for the seamless interaction between the human and the ubiquitous computing environment. Because of the physical proximity and pervasiveness of heterogeneous personal devices, sensors, actuators, and other devices and services, ubiquitous computing environments need a powerful infrastructure for coordinating accesses to these resources.

Gaia [RHC$^+$02], a part of the Active Spaces project at the University of Illinois, is a "meta" operating system that provides application developers with a uniform middleware abstraction to the ubiquitous computing environment. In addition, Gaia provides infrastructure services for naming and context, security services such as authentication, and access control [SNC02]. This provides a new paradigm of application programming. For example, sensors and actuators can detect and respond to contextual information such as a user's location in the environment, allowing the system to automatically configure the space to the user's preferences.

While ubiquitous computing systems are designed to boost the productivity of its users, the high level of coordination and control in the environment poses a threat to the privacy of its users. For example, the system can be used to track a user's movements throughout the ubiquitous computing environment, exposing the location privacy of its users. Furthermore, various services store audit-log information, which can be correlated to further expose the privacy of a user's system usage patterns. The recently built Siebel Center for Computer Science at the University of Illinois is an example of a ubiquitous computing environment, where the physical security of spaces is provided by electronic card readers, lights are controlled by motion sensors, doors are fitted with sensors to register "open" and "close" events, video cameras monitor public spaces, and so on. It is easy to see the privacy implications of such a system, which in effect is a distributed surveillance system if the right precautions are not taken. To provide users with privacy, proper care must be taken with the storage and dissemination of audit information. Furthermore, since ubiquitous computing environments such as Siebel Center are controlled by a single organization, care must be taken so that individual administrators in the organization cannot expose the privacy of the system's users.

We address the problem of user privacy in ubiquitous environments at three levels: location privacy and trustworthy communication, unlinkability of access transactions, and policy protection for access control feedback. In earlier work, we proposed a routing infrastructure for ubiquitous computing environments called *Mist* [AMCK$^+$02] to provide users with location privacy. Using *Mist* a user can access services in the ubiquitous computing environment through a remote proxy called a *Lighthouse*. Lighthouses were arranged in a hierarchy such that the Lighthouse chosen represented a tradeoff between the granularity of location privacy and the latency of communication. *Mist* was the first comprehensive system built for ubiquitous computing environments that addressed location privacy. In our efforts to improve *Mist* by replacing the static hierarchy of Lighthouses with a distributed network of heterogeneous routing elements, we realized that applying anonymous routing solutions such as Crowds [RR98] and Onion [RSG98] were inadequate within an organizational setting. While these solutions may be effective in widely distributed environments such as the Internet where routers can be assumed to be independent entities, at the scale of an organization they fail to provide protection against administrators who may have access to several, if not all, routers participating in the anonymizing network. At such scales, trust relationships must be exposed to users.

Our main observation was that routing elements in a heterogeneous network will have several attributes such as domain ID, administrator, physical security, OS version, and attack history. In Chapter 4 we present a trustworthy communication model that allows users to specify richer privacy requirements than the one-dimensional (quantitative) anonymity provided by other solutions. This provides users with more flexibility for their individual privacy demands based on perceived threat. For example, a user may choose to exclude routers from a particular domain, include only those routers with certified high physical security, exclude intermediate wireless links, and optimize paths based on quantitative metrics of trust, which we call "confidence." One of the main contributions of this thesis is a formal model that includes a representation of the network elements and their attributes, a language for specifying qualitative discretionary privacy policies of communication paths, a quantitative model for representing threat or trust, and efficient algorithms for computing paths based on these qualitative and quantitative specifications. We call this model "Routing with Confidence" and explore the boundaries of what demands are feasible for trustworthy routing and show that several demands of interest are computationally hard. We contribute several NP-hardness results in this regard.

The next aspect of privacy that we look at is the unlinkability of access transactions by users in a ubiquitous computing environment. Using our model of secure routing, a user can be confident about the privacy of his or her individual accesses to services. However, anonymizing and secure routing protocols are not sufficient for providing "unlinkability." Using timing information, for example, multiple anonymous accesses to various services can be linked to the same user, even if the user's identity is unknown. Suitable semantic analysis of the linked transactions could be used to expose the actual identity of the user. In Chapter 5 we present a model for achieving unlinkability through access control. Unlike other approaches based on cryptography or traditional separation of duty policies, we address the problem of restricting access to *related* audit logs or their replicas without requiring express coordination between the distributed audit log databases. These audit logs may require the storage of linkable information, and cryptographic mechanisms would be too restrictive or cumbersome. We evaluate our approach by proving security and precision properties of our access control model and show how versioning can be used to trade precision for evolution in protection state. We also evaluate the computational complexity of our approach.

Finally, in Chapter 6 we address privacy-preserving feedback about access control policies. At various points in ubiquitous computing environments, and our proposed models, users must present credentials to the system to gain access to resources. However, access control in ubiquitous computing environments is especially tricky because policies include contextual information. Having access to resources depends not only on the user's set of credentials, but also on context variables in the system such as temperature, and the presence of other users in the room. In such a system, it becomes necessary to provide users with feedback if access to a resource is denied. This promotes a more usable system, leading to stronger overall security. However, giving unrestricted feedback to users about system security policies can adversely affect the privacy of other users in the system. For example, telling a malicious outsider that only members of the CIA may use a particular room violates the privacy of CIA operatives. Providing users with useful and authorized feedback on access control decisions has not been addressed in the past. Our framework called *Know*, uses "meta-policies" to control feedback based on the user's credentials, and cost functions to rate feedback based on how useful it is to the user. We evaluate two specific cost functions and the complexity of our approach.

In summary, this thesis addresses the privacy demands of users in ubiquitous computing environments by providing qualitative and quantitative models that allow users and administrators to tune privacy parameters within the system based on perceived threat.

2 Background and Related Work

In this chapter we present background material on ubiquitous computing environments and privacy terminology. We then address related work in security and privacy aware routing protocols, unlinkability, and policy protection.

2.1 Ubiquitous computing

The goal of ubiquitous computing is to blend the computational or virtual environment into the everyday physical environment and to provide a mechanism for the seamless interaction between the human and the ubiquitous computing environment thereby boosting the productivity of its users. Because of the physical proximity and pervasiveness of heterogeneous personal devices, sensors, actuators, and other devices and services, ubiquitous computing is also referred to as *pervasive computing*. Ubiquitous computing is a broad concept, encompassing mobile phone applications interacting with various services over the phone network, the use of personal digital assistants (PDAs) at supermarkets to get information about products advertised through RFID, smart spaces or smart rooms that feature space applications to cater to users or collaborative groups, and so on.

The Active Spaces project at the University of Illinois is an example of smart spaces in ubiquitous computing [RHC$^+$02]. This thesis focuses on ubiquitous computing environments of this nature. Other examples of such environments include MIT's Project Oxygen [oxy], Carnegie Melon University's Project Aura [GSSS02], and Stanford's Interactive Workspaces Project [JFW02b]. We now present the relevant properties of these ubiquitous computing environments (with specific references to Gaia), which we will broadly refer to as smart spaces.

2.1.1 Smart spaces

We focus on ubiquitous computing environments that aim to provide a software infrastructure for managing devices, sensors, actuators, and other services for physically bounded regions such as offices, homes, and conference rooms. Smart spaces are ubiquitous computing “habitats” meant to aid users in a tightly-knit physical and virtual environment. We will refer to smart spaces as “active spaces.”

Likewise, we focus on organizational settings, where a “smart building” such as the Siebel Center for Computer Science at the University of Illinois is a federation of active spaces, also referred to as “super spaces” [AMCRC04].

2.1.2 Meta-operating system

Since active and super spaces are composed of heterogeneous services and devices (including embedded devices) with various operating systems, a uniform middleware abstraction enables the seamless integration of such devices and services. Gaia OS [RHC$^+$02] is an example of a middleware “meta-operating system,” which can be installed on a multitude of devices and their operating systems, and allows their integration into the active space. Like a traditional operating system for a single computer, Gaia allows application developers to view an active space as a single programmable entity with uniform APIs (Application Programming Interfaces).

Gaia manages resources within an active space and provides services for naming, location, context, event management, service discovery, and a context-aware file system.

2.1.3 Context

Central to the idea of ubiquitous computing is the use of contextual information to drive applications. For example, the current activity in the room can offer hints to applications, which can tune their behavior to the specific events in the active space. Location information can be used to detect users in an active space, and sensors can detect temperature changes. Gaia can use this context to react to changes in the environment, and ultimately react to the needs of its users.

One of the key issues that we discuss in this thesis relates to access control policies based on context variables. Such access control policies offer new challenges in ubiquitous computing environments, where access depends on context in addition to credentials.

2.1.4 Infrastructure

Ubiquitous computing environments feature a meta-operating system along with several infrastructure services. In an organization, applications can assume a uniform abstraction for communication, coordination, access control, and data storage. In terms of security, applications can specify quality of protection for communication, security policies can be uniformly enforced across the organization, and it is assumed that security can be tightly managed in such a distributed setting. This thesis provides models for security in such environments. In contrast, traditional distributed systems such as organizational networks cannot rely on a common set of security protocols in an organization. For example, different research groups in a department manage their own local networks, which are part of the larger departmental network. Attaching security policies to data is not sufficient for access control since other machines on the network are not guaranteed to enforce these policies. Active or super spaces give us the ability to control resource accesses throughout the organization, while retaining local control for individual space administrators. We will build upon and rely on this infrastructure in the models we propose.

2.2 Privacy

There are several notions of privacy in computing environments. We now introduce the standard properties of privacy and their definitions as stated in [P$^+$04], which includes detailed explanations of each property. Consider a communication network with the set of senders S, receivers R, and messages M. Senders and receivers have locations in L.

2.2.1 Anonymity

Definition 1. Anonymity *is the state of being not identifiable within a set of subjects, the* **anonymity set**.

The type of anonymity depends on the particular situation.

Sender Anonymity: A sender $s \in S$ may be anonymous within the set of potential senders S.

Receiver Anonymity: A receiver $r \in R$ may be anonymous within the set of potential receivers R.

We introduce the notion of location anonymity or location privacy.

Location Anonymity: The location of a subject (sender or receiver) is anonymous within the set of potential locations L. We also refer to this property as *Location Privacy*.

2.2.2 Unlinkability

Definition 2. Unlinkability *of two or more "Items of Interest (IOIs)" (e.g., subjects, messages, events, actions, etc.) means that within this system, these items are no more and no less related than they were related considering a-priori knowledge.*

For example, anonymity can be defined in terms of unlinkability. Here the IOIs are messages and subjects.

Sender Anonymity: A particular message m is not linkable to any sender, and that to a particular sender s, no message is linkable.

Receiver Anonymity: A particular message m is not linkable to any receiver, and that to a particular receiver s, no message is linkable.

Location Anonymity: Neither the identity of a particular subject, nor any message can be linkable with the location of the subject.

Sender-receiver unlinkability: If two parties a and b are communicating with each other, then it cannot be deduced that a and b are communicating with each other. This is also referred to as "anonymous connections" in the literature.

2.2.3 Unobservability

Definition 3. Unobservability *is the state of IOIs being indistinguishable from any IOI at all.*

This implies that messages are not discernible from random noise.

Sender Unobservability: It is not noticeable whether any sender within the unobservability set S sends.

Receiver Unobservability: It is not noticeable whether any receiver within the unobservability set S receives.

We discuss several protocols that provide anonymity and location privacy for subjects. However, sufficiently powerful attackers may be able to observe when a particular sender is sending messages. Likewise, if the attacker can observe a receiver receiving messages, the timing correlation might expose the fact that a sender and receiver are communicating with each other, exposing their privacy. To hide the fact that the senders and receivers are sending and receiving messages at all is usually accomplished by additional *cover traffic*, also called *dummy traffic*. If all senders and receivers send messages (legitimate traffic and cover traffic) at regular intervals, the attackers are not able to distinguish legitimate messages from dummy messages. Such approaches add the unobservability property to anonymous routing mechanisms.

2.2.4 Pseudonymity

Definition 4. *Being* **pseudonymous** *is the state of using a pseudonym as the identity.* **Pseudonymity** *is the use of pseudonyms as IDs.*

Pseudonyms are identifiers of subjects. For example, Alice may use the pseudonym "WonderWoman" and hence Alice's true identity is not readily identifiable within the system.

Pseudonymity is used in systems where users are given a limited form of anonymity by keeping their actions and pseudonym linkable to their identity. For example, if WonderWoman accesses two different resources, the audit logs can be used to link these two accesses. Furthermore, Alice's true identity can be linked to her pseudonym in cases of system abuse or legal subpoenas. Hence pseudonymity is an important means to offering privacy to users, yet retaining some degree of accountability of its users' actions.

Examples of pseudonymity include email addresses issued by companies such as Hotmail [hot], which require a credit card to establish an email account (pseudonym). In case of abuse, the user can be held accountable for his/her actions.

In addition to these four properties defined in [P^{+}04], we now discuss the more traditional notion of *confidentiality* of personal information.

2.2.5 Confidentiality

In security, **confidentiality** refers to the concealment of information or resources [Bis03]. For example, a company may want to protect trade secrets from its competitors, or a group of users may want to work on a document that cannot be read by people outside the group. Gaining such confidentiality for information can be achieved in a variety of ways. A security *policy* specifies the confidentiality requirements, and suitable mechanisms such as *access control* or cryptography can be used to ensure the confidentiality of data.

In the context of privacy, users are interested in concealing *personal* information such as their social security number, credit card information, location information, and what resources they can access. A ubiquitous computing environment is pervasive by definition and personal information about users is gathered throughout the environment. Proper care must be taken to maintain the privacy of the user by keeping personal information confidential.

2.2.6 Degree of anonymity

Reiter et al. [RR98] introduce terms that address the degree of anonymity between sender-unobservability and *provably exposed*, i.e., when a sender is known to be the originator. For example, the definitions presented for unlinkability assume a stronger notion of anonymity, where two IOIs are "no more or no less" related than they were a-priori. However, this notion of *indistinguishability* may be too strong. In certain cases, it may be sufficient to create "reasonable doubt" about the IOIs. Along these lines, Reiter et al. use the following terminology for sender-anonymity: *beyond suspicion* means that the sender is no more likely to be the sender of an observed message than any other sender in S. In other words, the probability that any $s \in S$ is the sender is $\frac{1}{|S|}$. *Probable innocence* is when a sender s appears to be no more likely to be the originator than to not be the originator. In other words, the probability that s is the sender is less than $\frac{1}{2}$. And finally *possible innocence* refers to the case when there is a non-trivial probability that a sender s is not the originator.

2.2.7 Threat

Definition 5. *A* **threat** *is a potential violation of a user's privacy.*

For example, the fact that administrators can access Alice's personal information is a threat to her privacy with respect to confidentiality. Administrators that can piece together router logs to expose Alice's communication path is a threat to her anonymity. These threats do not have to be realized since they are *potential* violations of privacy. Actual violations are called **attacks**. In this thesis, we will present models that allow users to specify privacy policies based on their own perceived threats in the system. Indeed, Alice may consider an unfriendly administrator Eve to be a threat to her privacy, but Charlie, who is Eve's close friend, may not. Hence, in the remainder of this thesis, it is important to keep in mind that each user will have different notions of threats to their privacy, and adequate models are needed to express such threats, and prevent attacks on a user's privacy.

2.3 Protocols for communication privacy

While very little attention has been paid to communication privacy in ubiquitous computing environments, there has been a substantial amount of research on anonymous communication for the Internet. These protocols aim to provide sender-anonymity, and some approaches also attempt to provide sender-unobservability through cover traffic. We discuss some of the important work in Internet anonymous routing such as Crowds [RR98], Mix networks [Cha81] and Onion [RSG98] routing, and then introduce *Mist* [AMCK$^+$02], which we developed at the University of Illinois specifically for location privacy in ubiquitous computing environments.

2.3.1 Crowds

Even though *Crowds* [RR98] was originally designed for the anonymity of web transactions, the underlying principle is applicable to anonymous routing in general. Consider the set of senders S, also referred to as a "crowd." Messages originating from a sender s are forwarded randomly within the crowd and eventually forwarded to the receiver. From the receiver's point of view, each sender is equally likely to be the originator of the message, giving the sender a high degree of anonymity, i.e., the sender-anonymity is "beyond suspicion."

In Crowds, each participant is called a *jondo*. The originator of a message picks a jondo from the crowd (possibly itself) and forwards the request to the selected jondo. At each stage, the request is propagated within the crowd with probability p or sent directly to the server (receiver) with probability $1 - p$. The parameter p is assumed to be constant within the crowd, and influences the average path length between the sender and receiver. Using this simple forwarding scheme, one can show that senders have sender-anonymity beyond suspicion with respect to the receivers. Furthermore, Reiter et al. show that the sender has probable innocence against malicious collaborating jondos for sufficiently large crowd sizes. This is proved using the parameter c, which is the fraction of compromised nodes in the crowd. As we will argue later, this assumption may be reasonable in widely dispersed crowds where routers can be assumed to be independent entities, but does not necessarily hold for crowds in smaller geographic locations, e.g., within a building.

2.3.2 Mixes

Chaum [Cha81] proposed an anonymous remailing scheme based on the concept of "mixing." Mail relays in the Mix network would receive emails, and reorder outgoing emails to break the association between incoming and outgoing emails. This was meant to foil traffic analysis by adversaries observing traffic entering and leaving the mail relays. Email messages were encrypted several times to encode the path of remailers. We discuss this technique in more detail in the next section on *onion routing*. The main contribution of Mix networks was to provide a scheme for sender-anonymity that aimed to resist traffic analysis.

2.3.3 Onion

Reed et al. [RSG98] describe an *onion routing* system meant to establish two-way anonymous communication channels. An "onion packet" contains a message using several layers of encryption. Each router that receives an onion packet can decrypt (or "peel off") the outermost layer of encryption, yielding the identity of the next router in the path chosen by the sender, and an onion packet to be forwarded to the next router containing the rest of the path. The main idea is that each router only knows the previous and next routers of a communication path, providing an anonymous connection between the sender and receiver. Onion routing is an important building block for establishing secure routes of communication since the routers along the path are oblivious to the sending and receiving parties.

Camenisch and Lysyanskaya [CL05] give a formal description of onion routing and provide a provably secure scheme for onion routing. They provide a scheme that satisfies the following properties:

Onion-correctness: If an onion packet is formed correctly and if the correct routers process it in the correct order, then the correct message is received by the last router.

Onion-integrity: Even for an onion created by an adversary, the path is of length at most N, where N is a well-known parameter in the network.

Onion-security: This property intuitively states that consecutive adversarial routers must eventually deliver a valid onion to an honest router along the path if there is one, and not learn the contents of the message. Otherwise, if there are no honest routers along the path, then the receiver is also adversarial, and the adversarial routers are assumed to have access to the message.

Based on these properties, Camenisch and Lysyanskaya provide an onion routing scheme that satisfies these properties. We will assume a secure onion routing scheme in the rest of this thesis.

2.3.4 Other protocols

The approaches we presented above were application layer solutions. For example, Chaum's Mixes addressed email services and Crowds addressed web transactions, Tarzan [FM02], however, proposed a system at the IP-layer such that anonymous senders could interact with any Internet related service. Routing through the Tarzan network terminated at a NAT (network address translator) to bridge between Tarzan routers and regular Internet hosts. Tarzan includes an efficient cover traffic scheme to prevent traffic analysis, and is a peer-to-peer approach like Crowds. Systems such as Onion and Chaum's Mixes suffer from the drawback that the first node contacted in the anonymizing network knows the originator of the message, by virtue of the originator not being part of the anonymizing network. Assuming a peer-to-peer environment solves this problem.

2.4 Mist

The protocols mentioned above were mainly focused towards providing sender and/or receiver anonymity. It is assumed that the parties communicating want to do so anonymously without revealing their identities, where the location and identity of the person were considered "synonymous."

2.4.1 Location privacy

The main contribution of *Mist* [AMCK$^+$02], developed at the University of Illinois, was to recognize and separate the two pieces of identity. In ubiquitous environments, users want to interact with services and other users with their disclosed identities, e.g., Alice and Bob, and are mainly concerned about their location privacy. Hence a system is required whereby users are available (or reachable) for communication in the system while their locations are hidden. How can one provide such a service to its users? The following research on *Mist* was conducted in collaboration with Jalal Al-Muhtadi, Prasad Naldurg, Luke St. Clair, and Seung Yi.

2.4.2 Lighthouses

To solve the location privacy problem and allow users to be reachable for communication, *Mist* introduced the concept of *Lighthouses* that served as communication points for users. This is similar in concept to the NAT endpoints in Tarzan. For example, Alice can register with a Lighthouse through an anonymous channel and make herself available for communication "in the mist." When Bob wants to contact Alice, Bob can communicate with her through her Lighthouse. Like with Onion, each *Mist Router* along the path from Alice to her Lighthouse only knows the previous and next hops. Hence, Alice's Lighthouse can route traffic to Alice without knowing her location. In effect, Alice chooses not to have sender-anonymity, but instead location anonymity (or location privacy). Note that Alice can choose not to disclose her name, and and use a pseudonym to register with her Lighthouse. In this case, Alice will have pseudonymity in addition to location privacy.

2.4.3 Hierarchy of routers

Mist Routers are arranged in a hierarchy based on geographic location. For example, Mist Routers corresponding to rooms will be the leaves of the routing tree, with floor Mist Routers one level higher, followed by departmental routers, and so on. A user Alice registers with a leaf Mist Router (a *Portal*). With the use of a carefully constructed packet, Alice can register with a Lighthouse "above" her Portal. Depending on her privacy needs, she can register with a Lighthouse higher or lower in the hierarchy, giving her more or less privacy respectively. In effect, Alice can pick the granularity of the geographic region in which her location is advertised. However, registering with a Lighthouse higher in the hierarchy will mean more hops of communication and higher latency in her routing, trading off communication efficiency for privacy.

The advantages of this approach are that the hierarchy is a simple and intuitive representation of spaces within a ubiquitous environment. Maintaining this hierarchy does not require special algorithms to maintain connectivity since the hierarchy is static. However, this also means that the routing infrastructure is not resilient to node or link failures, which can break the hierarchy and hence the routing protocol. Furthermore, all users who pick a certain region, e.g., First Floor, will impose a bottleneck at the First Floor Lighthouse. Lastly, Portals are inherently aware of the initiators of communication and can collude with Lighthouses to link the identity and location of a user.

2.4.4 A distributed approach

To tackle these problems, we proposed a distributed version of *Mist*. The main idea was to retain the idea of Lighthouses and to use an Onion like approach for selecting routers in the path to the Lighthouse corresponding to a geographic region. We assume that every participant is aware of the pool of available routers and their geographic locations. Furthermore, using a peer-to-peer approach solves the problem of forcing users to connect through portals. To give an example in the Internet setting, a user that wants to be hidden in the state of Illinois can pick routers that are within Illinois, whereas a user that wants a higher level of privacy can choose routers within the United States of America. The understanding is that a higher level of privacy will result in higher communication latencies, thus trading latency for privacy. There are several assumptions here: what if the routers selected within the United States of America are all within the city of Urbana? It is possible to have *much less* privacy than originally intended. This influences path selection. Picking routers uniformly at random from a set of routers is not sufficient. Now consider ubiquitous environments such as smart buildings. Picking five routers on the same floor may be scattered widely at the floor level, but might still be administered by the same person. Users should be able to constrain router selection and avoid certain routers if those routers are not "trusted." What should the user base trust on? What language can the user use to specify properties of paths? Are paths that satisfy such properties feasible to calculate? These are the questions that inspired our work in Chapter 4, which addresses the topic of "routing with confidence."

2.5 Trustworthy computing

"Trust" relates to the degree to which a user believes a certain property to be true. In security, we are interested in security properties such as resilience to attack – "can I trust this system to be virus-free?" or certificate validation – Alice may trust Bob's digital certificate of identity issued by Verisign [ver], but not one issued by some other user Charlie. As defined in [Bis03], an entity is *trustworthy* if there is sufficient credible evidence leading one to believe that the system will meet a set of given requirements. *Trust* is a measure of trustworthiness, relying on the evidence provided.

If trust is a numerical measure of trustworthiness, deeming a system as trustworthy may be based on a threshold of some trust value, or simply the system with the highest trust value is deemed to be most trustworthy. In this thesis we refer to this quantitative notion of trust as "confidence" or "diffidence" as the case may be. In some cases it is useful to measure positive attributes of trust, e.g., reputation based systems. We refer to these positive measures as “confidence.” Likewise, it is also useful to collect negative reports on behavior, e.g., intrusion detection reports. We refer to these values as “diffidence” values. We will refer to the trustworthiness of a route as its “Quality of Protection (QoP).”

We now discuss some general approaches towards trustworthy networking. Yi et al. [YNK01] propose the notion of *secure routing* for ad-hoc military environments. While their work focused on ad-hoc wireless routing environments and the specific credentials of the users and group key management, we present a generalized model based on different types of attributes of users and routers, and trust assumptions between these entities. For example, Yi et al. propose a mechanism whereby a certain group of individuals can encrypt data with the group key and avoid communicating through nodes outside the group. We present a generalized approach in which route selection can depend on several attributes and mathematical relations on links and routers. While Yi et al. perform route selection using broadcast messages, in our approach communication endpoints are given a graph of the network, and can compute routes based on policies that are not revealed to routers, even when the routers are part of the communication path. Finally, our model also incorporates a quantitative measure of trustworthiness of routes that are complementary to the qualitative routing policies based on attributes.

It is also worthwhile to address some of the research in multimedia, which attempts to find paths that satisfy “Quality of Service (QoS)” requirements, much like finding trustworthy routes with high QoP. Routing schemes have been proposed for some discretionary requirements such as bandwidth and latency. Resilient Overlay Networks (RON) [ABKM01] have been proposed to discover higher bandwidth (or lower latency) routes on the Internet by attempting to circumvent the standard underlying BGP policies. Selfish Routing [QYZS03] examines the effects of non-cooperative routing on the overall performance of the network and proposes algorithms that minimize the cost of selfish routing. Salsano and Veltri [SV02] describe a method to incorporate RSVP [ZDE^{+}93] in Policy Based Networks (PBN), where clients can specify QoS demands for their route. Constraint Based Routing (CBR) in Multiprotocol Label Switching (MPLS) [J^{+}02] allows clients to specify certain constraints (again, concentrating on QoS). The network then computes paths for the clients based on these constraints.

We address discretionary security requirements of users that desire a higher Quality of *Protection* (QoP) rather than a higher Quality of *Service* (QoS). Moreover, clients do not have the ability to keep their policies private in the QoS protocols, and must be disclosed to the network for admission control. We present a model that keeps the user's privacy policies secret from the routers.

Trust depends on the property in question. One of our motivating factors was privacy protocols such as Crowds and Onion mentioned earlier. Users may be interested in setting up routes that preserve their privacy, in terms of location anonymity or identity anonymity. A user may desire a trustworthy route that is likely to preserve the user's anonymity. Our model enables users to set up routes through routers in a way that does not compromise their privacy, leveraging on our experience with *Mist*. Users can specify trust attributes to avoid certain nodes and prefer some routes over others, rather than relying on the system to make anonymous routing decisions. This system resembles Onion routing, except that we address the route selection phase, which is done prior to route setup. As with *Mist*, this route will be used to connect to a Lighthouse and facilitate location privacy in ubiquitous computing environments, which allows users to be reachable through their Lighthouses.

2.6 Unlinkability of access transactions

Private access to services in a ubiquitous computing environment goes beyond anonymity. A user may want to ensure that multiple audit log records cannot be collectively analyzed to "link" the user's transactions together. A user may want this to be true even when anonymity is not required, so that an administrator cannot learn the user's access patterns. We now present related research in the context of ensuring unlinkability across different access transactions within a session. Research on unlinkability in the past has mostly focused on cryptographic mechanisms for anonymous authorization.

We first examine different cryptographic techniques that allow a user to disclose only those attributes that are strictly necessary for a given access transaction. One of the first proposals in this direction is the work by Brands [Bra00], where he proposes a certificate system that gives a user control over what is known about the attributes of his or her certificate (or authorizations), and can prove their possession using zero-knowledge protocols. However, with this scheme a user who presents the same certificate twice can be linked across his or her sessions with the same server, even though the attributes are still hidden.

Other researchers have explored the construction of credential systems that satisfy the *multi-show* property whereby the owner of a certificate can construct two or more credentials with the same attributes that are unlinkable[Ver01; PV03]. The construction of *anonymous credentials* presented by Chaum in [CE86] relies on interaction with a trusted third party for unlinkability. Camenisch, Lysyanskaya et al. [CL01; LRSW99] extend this unlinkability based on computational zero-knowledge proofs, and the credential system proposed in [PV03] defines what the authors call Chameleon certificates that provide a user complete control over the amount of information revealed as well as computational zero-knowledge proofs for unlinkability of credentials, provided these credentials can be encoded as linear Boolean formulas.

In Chapter 5 we argue that preserving unlinkability across access transactions using anonymous credentials can be hard to achieve. One of the issues with anonymous credentials is that although the identity of a user is not revealed by engaging in multiple access transactions, the list of attributes revealed at the end of the access negotiation can be logged by the server, along with timing information. Multiple transactions can be semantically correlated using this information. Furthermore, a system may not be able to support anonymous access transactions if required by law, or simply if accountability is desired.

As explained in Section 5.1, the unlinkability problem we define differs from traditional Separation of Duty (SoD) problem of preventing a single user from performing different actions on the same object [SZ97]. In our problem, we want to prevent an unauthorized user from accessing different audit records associated with different information flows initiated by a single user. This is similar to *conflict of interest* classes of information in Chinese Wall [BN89] policies. However, it is not feasible to impose centralized control or history based approaches to different objects and their replicas like with Chinese Wall policies or dynamic SoD. We see a decentralized approach to enforce unlinkability.

In their discussion on different types of SoD constraints for RBAC, Simon and Zurko [SZ97] distinguish between three types of SoD constraints : static, dynamic, and operational. Given a set of static SoD constraints, policy conformance reduces to checking if the roles involved have disjoint memberships so that no single person has access to all operations in a workflow.

With respect to enforcing dynamic SoD constraints, Sandhu's work on Transaction Control Expressions (TCE [San98]) shows how dynamic SoD constraints can be enforced adequately using history if the information about each transaction is annotated with the object itself. Simon and Zurko argue that such history is essential to enforce general SoD constraints. Gligor et al. [GGF98] formalize the relationship between SoD and RBAC and show how RBAC is not sufficient to enforce all types of SoD properties, especially dynamic SoD constraints. More recently, Li et al. [LBT04] show how directly enforcing static SoD policies is intractable, let alone dynamic SoD policies, and show how statically mutually exclusive roles can be engineered to enforce these constraints on a best-effort basis.

In the context of our unlinkability problem, annotating audit records in different databases with history information does not provide us a mechanism to enforce unlinkability as these data objects are independent and local history cannot be used to enforce global constraints. Minsky [Min04] proposes a decentralized approach for enforcing Chinese-Wall policies by processing accesses by an agent or a group of agents, and enforcing the Chinese-Wall policies locally, instead of a centralize reference monitor. However, the problem with such an approach is that some entity must maintain a history of accesses by the person or group. Instead, our proposed solution annotates different audit records with authorizations to enforce unlinkability in a decentralized setting, where each access can be allowed or disallowed based on a local decision without maintaining history.

In terms of detecting semantic conflicts that can be exploited by a user to correlate different types of audit records and expose the privacy of a user, a number of data mining techniques that explicitly represent knowledge can prove to be useful. Researchers have examined how to use data mining techniques to correlate logs in the context of intrusion detection to detect attacks [LS98; UJ03]. We believe that some of these techniques can be extended to look for unlinkability conflicts at the semantic level. As mentioned in Section 5.2, our framework examines the unlinkability problem at the level of authorizations to access audit flows. Analysis of whether two flows are linkable semantically can be leveraged to improve the precision of enforcement of unlinkability policies.

Finally, Hong [Hon05] presents the Confab toolkit for creating privacy aware applications. Policies can be attached to data to restrict the flow of private information in the system. Our work on unlinkability can augment the Confab framework by generating unlinkability policies for sensitive data in addition to simple confidentiality policies.

2.7 Policy protection

Finally, in Chapter 6 we address the issue of providing feedback about access control policies to users that are denied access to resources. We are not aware of any work that addresses the issue of providing useful feedback to users, while protecting the privacy of other users (i.e., what they can access) in the system.

Policy hashing [KHJ03] has been proposed to protect the policies for a firewall from less trustworthy enforcement points. This prevents intruders from reading sensitive policies on compromised enforcement points. Feedback to end-users is not a consideration. Access control systems for Web publishing [BDS01] provide more information about the policy if conditions needs to be changed for access. However, policy protection is not addressed. Trust negotiation protocols [BS02; WL04; YWS03] address the problem of protecting the confidentiality of credentials of both parties involved in a session. At each stage, both parties must satisfy each others policies to proceed with the negotiation. *Know* can augment these systems at each stage of trust negotiation by providing useful feedback. With respect to suppressing feedback options, Bonatti et al. [BS02] protect the server's *state* by filtering policy feedback. Such techniques can also be applied to *Know*, which protects the server's *policies*. Policy protection in [BS02] is achieved by progressively revealing more requirements depending on credentials revealed by the user.

3 Problem Statement and Thesis

Ubiquitous computing poses several new challenges for the privacy of its users. Users can seamlessly interact with a plethora of devices in a pervasive physical and virtual environment. Accesses to services and devices are controlled by a meta-operating system such as Gaia [Gai]. Some of these devices are sensors and actuators which can detect and respond to contextual information such as a user's location in the environment. While such functionality is designed to boost the productivity of users, this very design can be used to track users' movements through the ubiquitous environment. Users should be given the flexibility to specify privacy policies for their communication based on their perceived threat to privacy. Furthermore, various services store audit-log information, which can be correlated to further expose the privacy of users' system usage patterns. In effect, the environment becomes a distributed surveillance system, and proper care must be taken with the storage and dissemination of audit information. Users must be given the option to fine tune their privacy parameters in the system based on perceived threat, and be given suitable feedback about security decisions.

3.1 Problem statement

Ubiquitous computing has made it easier for an organization to track users' movements, communication, and accesses to services, thereby posing a threat to the privacy of its users. The privacy requirements of users in ubiquitous computing environments have not been adequately researched and sound models are needed to allow users to control and optimize their privacy requirements in such environments based on their individual perceived threats.

3.2 Thesis

The privacy demands of users in ubiquitous computing environments can be satisfied through the synergy of theoretical security models to expose potential threats to users, expressive policy specification languages based on qualitative and quantitative properties to express a *user's perceived threat, and efficient algorithms for enforcing privacy policies based on these models and specifications.*

This thesis addresses the following privacy issues:

3.2.1 Communication privacy and trustworthy routing

In Section 2.3 we discussed various protocols that aim to provide communication privacy properties such as sender/receiver anonymity and anonymous connections. The main drawback with such approaches is that they assume a uniform attack model, and treat all nodes equally. The user does not have the power to restrict or prefer nodes in the network based on trust relationships. Protocols such as Crowds [RR98] prove statistical anonymity, which has more credence in a widely distributed setting. Furthermore, these services focus on sender anonymity. These limitations, along with the lack of an infrastructure to separate identity from location, inspired us to develop *Mist* [AMCK$^+$02], a protocol for location privacy. Using *Mist*, users can access services with their regular system identities while keeping their locations hidden from these services. Our main contribution was to provide users with the facility to choose varying granularities for their advertised locations, trading communication efficiency for location privacy.

While various anonymous routing systems, including *Mist*, can be used to achieve location privacy, as argued above, users must be given the additional power to specify qualitative and quantitative constraints for their communication in more restricted settings such as within an organization. This will improve the ''quality of protection" (QoP) or trustworthiness of their routes used for location privacy. For example, all routers may be under the control of a single administrator. Providing an appropriate model that allows users to express discretionary security and privacy policies for trustworthy routing will allow users to communicate privately in environments that have traditionally specified network security policies of a mandatory nature. Since each individual has his or her own notion of privacy, a model for such a system should allow users to specify a rich set of privacy policies based on their own *perceived threat*, while keeping the algorithms to satisfy these policies efficient. Indeed a user may demand a path that is a solution to an NP-hard problem. Hence it is important to identify and study what models and policy languages allow for efficient trustworthy routing.

We present the results of our research in Chapter 4.

3.2.2 Audit-log unlinkability

After using suitable location privacy and secure routing mechanisms to hide the identity and/or location of a user, audit information of that user's accesses to various services is stored across various databases. It is possible for other users such as system administrators to correlate transaction information, including timing, across audit logs to expose identity, location, transaction history, and other sensitive attributes of a user. In Section 2.6 we discussed several approaches to this problem. Cryptographic approaches such as anonymous credentials suffer from the problem that audit records might still contain semantic information (e.g., timing) that allows the linking of records. Cryptographic approaches are not sufficient protection against these attacks. Traditional policies for separation of duty (SoD) apply to accesses to a single object. While it is easy to satisfy SoD constraints for a single object, it is difficult to regulate accesses to *related* objects and their replicas without history based approaches as with Chinese Wall [BN89] policies. Decentralized approaches for ensuring unlinkability of such records are needed, along with an analysis of the security properties provided by such models. Since each user will have different privacy requirements, the model must allow users to specify unlinkability polices based on their perceived threat to unlinkability. In this thesis we explore an approach based on access control for regulating access to audit records based on negotiated policies, and explore the properties that can be guaranteed by our model.

We present the results of our work in Chapter 5.

3.2.3 Privacy-preserving feedback

In Section 2.7 we discuss several techniques to hide policy authorizations. However, these approaches do not address the issue of providing users with useful feedback on access control decisions. In most cases, revealing policies is considered to be a security breach because this reveals too much information to the denied user, who is potentially malicious. Furthermore, access denials imply that users do not have sufficient credentials to access the resources. Ubiquitous computing environments add more challenges by basing system policy on contextual information. Access control decisions no longer depend on credentials alone. To avoid the confusion and frustration of users who could access resources previously (e.g., "just an hour ago"), a system of feedback is needed for its users. Furthermore, there are privacy implications of revealing too much information about the system's policies to its users. For example, a professor's permission set might be kept secret from students. Hence, a model for feedback should also include some form of policy protection to maintain the confidentiality of sensitive access permissions.

We present our model *Know* in Chapter 6, which addresses the issue of useful feedback and policy protection in ubiquitous computing environments.

3.3 Success criteria

I propose the following broad criteria for evaluating the thesis:

1. Does this thesis advance the state of the art for privacy models in ubiquitous computing environments?
2. Are the models proposed expressive enough?
3. Are the models proposed computationally efficient?
4. Do the models empower users and administrators to fine tune discretionary and mandatory privacy parameters in ubiquitous computing environments?

For each specific model, I propose the following criteria:

1. **Trustworthy routing**
   (a) Does the proposed model capture the diversity of trust relationships in the network?
   (b) Does the proposed model allow for expressive routing policies based on these relationships and perceived threat?
   (c) Is the policy language computationally easy to satisfy?
   (d) Can these models also support quantitative representations of trust?
   (e) Does the thesis evaluate feasible and infeasible representations of trust?
   (f) Can the model find solutions to both the qualitative and quantitative privacy policies efficiently?
2. **Unlinkability**
   (a) Does the proposed model capture the flow of audit information in a ubiquitous computing environment?
   (b) Can linkability conflicts be identified efficiently?
   (c) Does the proposed model allow users to restrict linkability of their private audit information based on perceived threat?

    (d) Is the solution decentralized and efficient to enforce?

    (e) Does the thesis prove the security of its proposed model?

3. **Feedback and policy protection**

    (a) Does the proposed model allow administrators to fine tune feedback disclosure? Are these methods expressive enough?

    (b) Does the proposed model allow for useful feedback? What are the approaches for providing users with relevant feedback?

    (c) Can feedback be computed easily? What techniques are used, and what are their advantages?

    (d) Does the proposed model make access control in ubiquitous computing environments more usable?

4 Routing with Confidence

In Chapter 3 we argued that unlike traditional approaches for communication privacy in the Internet, trust relationships need to be considered for communication privacy. In this chapter we present our model for trustworthy routing, which allows users to specify discretionary privacy policies based on their perceived threat. Trust relationships are exposed to the user in terms of attributes of links and routers. We begin this chapter with a brief overview of policy based networking (PBN), which we assume to be part of the ubiquitous computing environment.

4.1 Policy Based Networking

With the advent of policy based networking, network administrators now have the ability to specify, administer, and enforce an organization's network-access and utilization policies more effectively. PBN has traditionally focused on which users have access to what resources in a network [SL02]. A PBN framework uses bandwidth management, traffic-flow management, firewalling, caching, and other routing protocol and network security solutions such as IPSec, VPNs, etc., to provide differentiated services to groups of users in a dedicated network.

For most part, the policies in a PBN refer to *mandatory* access control (MAC) and utilization policies that the network, as a system, applies to its users. The PBN architecture [WSS$^+$01] organizes different network objects such as resources and services into different *object roles*, and defines a policy as a relationship between these object roles and different *user groups*. For example, traffic from certain groups of users can be treated preferentially, or access to certain network resources can be restricted to users belonging to a specific group. In addition, policies can be defined based on the attributes of the traffic itself—e.g., music file transfers or other application specific packets can be bandwidth-limited. PBN Policies are stored in a (possibly distributed) policy repository and enforced at Policy Enforcement Points (PEPs) on firewalls, routers and switches, etc. using a wide variety of mechanisms such as access control, filtering, and queue management.

The PBN framework has greatly simplified the management and administration of organizational network security policies. We extend this framework that incorporates a user's privacy expectations and preferences, with the existing mandatory network policies, to influence the path chosen by a user's traffic within this setting. Our motivation stems from the observation that the *discretionary* privacy demands of users have been largely ignored in any formulation of PBN policies, and for communication privacy in general.

Our initial work on trustworthy routing was published in [KNC04].

4.2 Approach

We introduce a model of the network as a labeled state-transition diagram, and use a subset of Constraint Linear Temporal Logic (CLTL) [DD02] based on integer periodicity constraints for discretionary policy specification. These formulas are based on attributes of entities in the network, which may be qualitative or quantitative. This approach allows us to specify qualitative communication path properties based on quantitative attributes, and explore algorithms to discover routes that satisfy users' policies. For example, a user may demand a path that visits only physically secure routers (a qualitative demand) with fewer than 15 intrusion reports (a quantitative demand) in the previous week. We incorporate a threat/trust model to rate routes based on their overall *confidence* levels. For example, attributes of routers maybe be true with a certain degree of confidence. The combination of these approaches allows users to set up routes of high confidence that satisfy their discretionary policies. We show how our policy language can be efficiently interpreted over the network model and be combined with shortest path algorithms for certain models of confidence. As described in Section 2.5, confidence or diffidence is a measure of trust for a node. The Quality of Protection (QoP) refers to the trustworthiness of a route. In Section 4.7 we define path-confidence, a measure of QoP.

4.3 Overview

Our proposed framework explicitly models static and dynamic trust attributes of both users and network objects and effectively captures the changing trust relationships between them as the system evolves over time. To illustrate, consider a user who may want to avoid certain routers based on the knowledge that the routers may be compromised because they are running outdated software with known security holes. The system administrator may not have installed the latest patch, or the patch may not be available. Note that the user's demands in this situation do not violate the mandatory system policy in any way. While a user would be dependent on the administrator in a traditional PBN, in our proposed model, a user can encode this requirement and discover a path dynamically, consisting of routers that do not have this vulnerability, and use only these routers until the vulnerability is patched.

Other examples of a user's discretionary policies in this setting include the ability to exclude routers that belong to an administrative domain that the user does not trust, or exclude routers that are dropping an unacceptable fraction of packets, and so on. A point to note here is that some attributes of both the user and the network object are dynamic, in the sense that they may change over time. We list different types of attributes of both user groups and network objects and classify them according to whether they are inherent, consensus based, or need to be inferred by the user in some way. This extends the traditional notion of "Quality of Service" to what we refer to broadly as the "Quality of Protection" (QoP) [CLM$^+$00; YNK02] of a network route. We explore the issue of trust management and describe what entities we need to enable certification and validation of dynamic trust attributes.

In order to capture the effect of dynamically changing trust relationships on the quality of routes our model can discover, we introduce a quantitative measure called *confidence*. Using this metric, we describe different functions to combine meaningfully the confidence values of individual links along a route quantitatively, presenting what we believe is a novel computational model of trust relationships. Confidence values also capture threat by changing the confidence levels in response to exposed threats and vulnerabilities. We show how we can efficiently compute routes that maximize the confidence a user can expect given the current threat model and trust relationships. We explore these issues in the context of three representative environments—a military network, a ubiquitous computing scenario, and a peer-to-peer network.

We envision a network in which users operate under the overall network MAC policy, but have the flexibility to apply dynamic trust attributes and relationships for improved security and privacy guarantees of their communication.

4.4 Assumptions

We assume a network for a ubiquitous computing environment in a single administrative domain such as a corporate or private network. Since these networks are effectively isolated from the Internet at large, they provide adequate support to en-

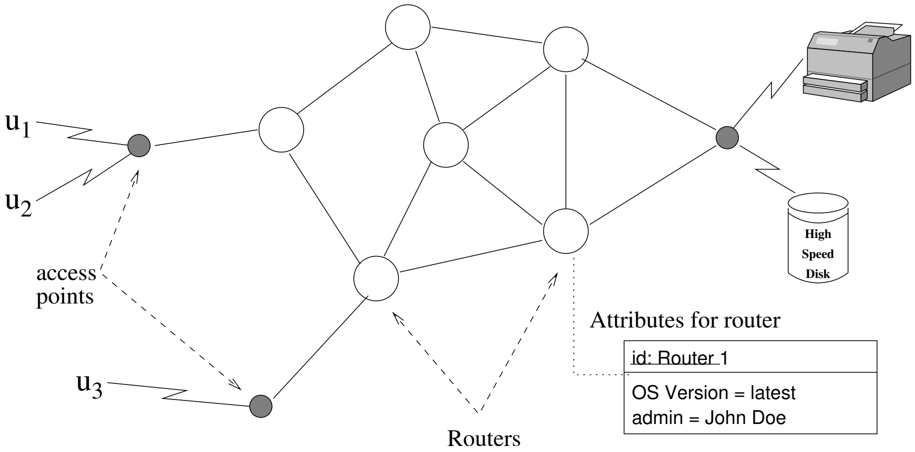


Figure 4.1: Architecture Overview

force cohesive administrative and management policies across the network. This allows for the use of policy based networking and specialized protocols for "high-confidence"communication within the organization. We assume that attackers can actively drop, modify, or inject packets into the network we assume the use of end-to-end encryption to detect such activity. We assume a suitable Public Key Infrastructure (PKI) and centralized or distributed trust authority for issuing certificates for *attributes*.

4.5 Solution technique

In this section, we present a high-level architectural view of our proposed model consisting of different network elements. Similar to traditional PBNs, our network includes a policy database, PEPs, and PDPs. Within our network we also have the ability to certify different static and dynamic attributes of users and network objects, through the means of a centralized or distributed trust authority.

As shown in Figure 4.1, users connect to our routing infrastructure through *access points*. Services can be connected to access points as shown, or certain services may be available at the routing nodes itself (e.g., discovery services that are part of the routing infrastructure). Based on the certified attributes that the user chooses to disclose to the authenticating system (for privacy reasons the user may only disclose a subset of their current attributes), the user is presented with a snapshot of our system consisting of different network elements, including routers, links and servers. Note that this snapshot is a restricted view of the network, reflecting what resources a user is authorized to use based on the user's disclosed credentials [1],

[1] We use *credentials* and *attributes* interchangeably since attributes are certified and are presented

according to the *mandatory* access policies of the organization.

The user hence possesses a logical view of the routers, their attributes, and their connectivity. When a user wishes to communicate with another entity on the network, he or she looks up the access point of the destination and computes a route to that access point. Within this view of the network, our framework allows the user to restrict their preferences for services and routes even further, in accordance with their discretionary demands. In the next few subsections we describe how each part of this process works, along with the trust negotiation and bootstrapping that occurs in the system. We begin with how attributes can be certified in our proposed system.

4.5.1 Attributes

Our first result was to define three types of attributes to capture both the static and dynamic nature of evolving trust relationships in our system—*inherent attributes*, *consensus-based attributes* and *inferred attributes*. Routers are associated with attribute-value pairs. As we show later, these attributes help us quantify the trust relationships in the system by associating them with a quantitative measure of "confidence."

Inherent attributes: These attributes are relatively static characteristics of an entity, which can be certified by a Certificate Authority (CA). Examples of inherent user attributes are identity, role, age, and gender. Inherent router attributes can include physical location, administrative authority, physical security, clearance level, and firewall security. A CA that can create attribute certificates and distribute them to users and routers. For example, users can use these attributes to set up routes through routers that are physically secure and that belong to a certain trusted administrative entity.

Consensus-based attributes: These attributes relate to the behavior of an entity with respect with other entities in the system. For example, routers in the network can vouch for the integrity of neighboring routers if they appear to be routing packets correctly. A compromised router may stop forwarding packets, and neighboring routers would degrade their trust in that router with respect to packet delivery. Users can therefore use these dynamic attributes to set up routes through routers that have been routing packets reliably on a need-to-use basis. Routers may decide that a certain user is not honoring routing policies and exclude that user from future negotiations. For example, the user may be running a transfer protocol that

as credentials

does not have any congestion control mechanism (e.g., non TCP-friendly multimedia flows). Hence routers may or may not vouch for a user's behavior, which would hurt the user's ability to set up future routes. This encourages good behavior of both users and routers within the network. Since these certificates are issued for the current behavior of a router or a user, it is impractical to have the CA issue such certificates.

Therefore, we need a robust and efficient protocol where routers and users can generate, agree, and distribute these relatively dynamic attributes. Since users and routers, especially compromised ones, can lie about these attributes, we suggest the use COCA [ZSvR02], an online certification authority that uses threshold cryptography to issue these certificates. The basic idea is that at least k out of n routers would need to agree on an attribute to issue a certificate for that attribute. COCA comes with built-in intrusion tolerance for Byzantine failures, and is reasonably efficient.

Inferred attributes: While entities in the network may have inherent or consensus-based attributes, users may have reasons not to trust certain routers, and likewise, certain routers may not trust certain users. For example a user might infer (through probes for example) that certain routers are running outdated versions of software with a known vulnerability. This is an indicator that the router may be compromised and is not trustworthy. Hence a user may want to avoid such routers. Since these are attributes that the user assigns to routers (or vice versa), these attributes are local to the entity making the inference. No certification is required for such attributes. Other examples include latest patches, daemons running, past behavior observed by the user, etc.

In the next subsection, we briefly explore how to accommodate for a user's privacy preferences.

4.5.2 Trust negotiation

In our system, we would like to honor the privacy of users. A user would like to reveal only those attribute certificates that are absolutely necessary to accomplish the user's goals. For example, a user may want to use the network as a *Student*, without revealing the actual identity. Since the logical view of the network depends on the credentials of the user, this view is restricted based on the attributes the user reveals to the network. Moreover, when a user demands consensus-based attributes of routers, the router may first demand that user present credentials appropriate to that demand. For example the router may disclose routing statistics only to users with a high level of security clearance (high-priority users). This suggests the use of trust negotiation protocols such as those proposed by Yu et al.[YWS03]. Such protocols can be effectively used to bootstrap trust between users and routers based on inherent and consensus based attributes.

In Chapter 6 we present *Know*, a model for providing users with authorized feedback for aiding usability in the authorization process. If a user is not satisfied with the authorized logical view of the network based on the current credentials, the system can inform the user which credentials might help gain access to other routers.

4.5.3 Routing model

Our next result formalizes the routing model and describes how users can specify their discretionary policies based on attributes of routers in the organization. As explained before, users can obtain a map of the network that they are authorized to view according to the organizational mandatory policy at startup. This map lists all the routers, and links, and labels each router with the set of static attributes that are valid on that router. Users can negotiate a larger map with the system using *Know* which is described in Chapter 6. Users are allowed to update this map with dynamic attributes at any point in time.

We model our network as a labeled state-transition diagram similar to Kripke structures used in model checking [CGP00]. Formally, a Kripke structure is the tuple $M = \langle S, S_0, R, L \rangle$, where S is a set of states, S_0 is the set of initial or start states, $R \subseteq S \times S$ is a transition relation between states, and $L : S \rightarrow \mathcal{P}(AP)$ is a labeling function where $\mathcal{P}(AP)$ is the power set of atomic propositions AP. Given a state $s \in S$, $L(s)$ is the set of atomic propositions that are true in s.

In the case of attribute-based routing, the set of routers corresponds to the set of states S in the model. If two routers s_1, s_2 are connected then $(s_1, s_2), (s_2, s_1) \in R$ since we assume symmetric links. Each relation in R corresponds to the connectivity between routers. The set of attributes at each router can be viewed as atomic propositions (or truth valued statements) about attributes in that that state. Therefore the set AP is the set of all possible attribute-value pairs in our system. For example, the attribute-value pair $a = \langle \mathit{OSVersion}, \mathit{4.0} \rangle$ is an atomic proposition that is *true* for routers with this specific attribute-value pair, i.e., OS Version 4.0. Without loss of generality, we will refer to attribute-value pairs as atomic propositions. In our previous example, the attribute-value pair $\langle \mathit{OSVersion}, \mathit{4.0} \rangle$ is represented as the atomic proposition a. Note, this set is finite in our model. The set of start states S_0 are specified by the user. We present our full network model in Section 4.6.4

after introducing link attributes and variables.

The user can now define their discretionary policies as path characteristics using temporal logic formulas that can be interpreted over what is called a computation tree of a Kripke structure. Formally, an infinite computation tree is obtained by unwinding the state-transition graph by starting with a fixed start state and applying all transitions from that state to other states in the model, and so on. For our purposes, we only consider finite computations, or in other words, finite paths.

Different types of temporal logic have been studied extensively in the past [CGP00] to describe properties of these infinite computation trees. We believe that the most useful logic for our case is Constraint Linear Temporal Logic (CLTL), which is used specify characteristics of paths in this tree. In addition to standard LTL, our proposed fragment of CLTL can express quantitative properties of paths. We do not define the syntax and semantics of LTL as it is well known, but explain how we can use it with quantitative constraints to specify properties in the next section.

One of our motivations for using this formalism is the availability of automatic tools that can compute efficiently whether there exists a path in our model that satisfies the constraints imposed by the LTL formula. This process is called model checking. In general, a model checker provides a counter-example (if one exists) to a property specified by the user. Specifying the negation of a desired property yields a path (counter-example) with the desired property. Model checkers can be modified to return more than one counter example to yield all paths that satisfy a specific type of LTL formula [SJW02]. While this approach can be computationally expensive, in the next section we show how we can adapt this technique to a computationally inexpensive subset of CLTL and highlight specific characteristics of our problem that make it particularly scalable.

Link attributes

We augment this model to also represent link attributes. We overload the definition of L to include the function $L : S \times S \rightarrow 2^{AP}$ maps a link (u, v) to its set of attributes $L(u, v)$. The resulting model is now a labeled state-transition diagram with a labeling function over nodes and edges.

4.6 Path specification

LTL formulas are a powerful way for users to express qualitative path requirements. As explained in the previous section, model checkers can be used to generate multiple paths, when they exist, that satisfy these constraints between a source access point and a destination access point in our model. Model checking algorithms for LTL formulas in general have time complexity $O(|M|2^{O(|f|)})$, where $|f|$ is the size of the LTL formula.

In addition to exponential dependence on $|f|$, traditional model checking can also be encumbered by large state spaces (large $|M|$). Some systems with simple high level specifications may result in a "state space explosion." For example, a state transition occurs in a Kripke model when the truth values of the atomic propositions in that state change. As a result, computation trees that represent all possible behaviors of the system by enumerating states and transitions for all combinations of changes of these values can become very large. Unlike such systems, the state space explosion problem is not a concern for us since $|M|$ is the size of the network.

In our case, the attribute certificates are fixed for a particular view of the network. We do not model the changing values of these attributes as different states for each router. Hence our approach is an "online" approach where we represent the *current* state of routers in the network as opposed to all possible states each router can be in at any given time. The transitions can only occur between routers that have links between them in the real network we are modeling. Therefore, we can limit the size of our model $|M|$ by number of routers, links, and attributes in our network, and we are only limited by the complexity of algorithms for verifying LTL formulas.

While the overall complexity is low for smaller LTL formulas, finding paths satisfying longer LTL formulas can easily become prohibitively expensive because of the $2^{O(|f|)})$ term. Since we augment this model with quantitative confidence metrics, finding paths of highest confidence that satisfy the LTL formula becomes even more challenging and instead we focus on a fragment of Constraint LTL for which the complexity of finding satisfying paths is linear in $|f|$. In addition, CLTL allows users to specify policies using quantitative variables. In effect, our model will reduce to running shortest path algorithms on a directed graph of size $|M|$ after some inexpensive transformations on the graph. We present our policy language in Section 4.6.5

Manna and Pnueli [MP92] define three useful classes of properties of paths that can be represented as temporal logic formulas—Invariance, Response, and Precedence. Invariance properties are true in every state in a path. These properties are useful to model user constraints such as "Route through nodes that support IPSec only". Response properties are useful to model quantitative properties of bidirectional paths, e.g., in terms of round trip latency or available bandwidth. Precedence properties capture the causal relationships between properties along a path. We explore these properties in turn and show they can be specified in LTL. We present the use of CLTL in Section 4.6.5 where we discuss quantitative representations of trust.

4.6.1 Global or invariance properties

Consider LTL formulas of the form $\mathbf{G}\,p$. $\mathbf{G}$ is the "globally" operator which means that in all states and links along the path, proposition p must evaluate to true. We restrict p to propositional formulas — users specify boolean formulas with respect to the attributes. The user requires that p must hold at each individual router or link. The algorithm for computing paths that satisfies $\mathbf{G}\,p$ first eliminates all nodes and links from the graph (state-transition diagram) where p does not hold. This solely depends on attributes at each router or link, and attributes at one router or link do not affect the satisfiability of p at another. The graph that we are left with represents the routers that the user is willing to route through.

4.6.2 Response properties

These properties are of the form $\mathbf{G}(p \rightarrow \mathbf{F}q)$ where $\mathbf{F}$ is the "finally" operator. The formula asserts that it is always true on our path that if proposition p is satisfied at any node, eventually proposition q will be satisfied. This property is useful to specify bounded-response and causal relationships between attributes. Quantitative versions of these properties (obtained by augmenting both the model and the temporal logic carefully with time variables as in [AH92]) can be used to specify path latencies and bandwidth constraints. We examine causal relationships of attributes with respect to precedence properties as described next.

4.6.3 Link and precedence properties

Next, we look at the case when certain attributes along the path must occur in a specific order at routers. For example, the user may want to set up a path that goes through routers in a non-decreasing order of classification levels. Once a packet enters a router with high level of security, it must not pass through a node with lower security. Consider the case when routers append sensitive information to packets. If the packet is at a certain router, it can never contain previous data from a higher clearance router, and hence there is no information leakage. The user can specify an attribute ordering $p_1, \ldots, p_n$, where exactly one of these is true at every router.

If p_i and p_j occur along a path, it must be the case that p_i occurred before p_j. In traditional LTL with only router attributes, this would be specified with the formula: $\neg \bigvee_{i>j} \mathbf{F}(p_i \rightarrow \mathbf{F}p_j)$. We show how this can be represented as a global property on links. We define the "Global Link Operator" $\mathbf{G_l}\, p$, where p is a propositional formula applied only to links. Consequently, we replace the previous $\mathbf{G}$ global operator with $\mathbf{G_r}$, which applies to properties of routers. For $\mathbf{G_l}$ we allow past and next $\mathbf{P}$ and $\mathbf{X}$ operators for specifying relationships between the endpoints of a link. Hence we focus on the class of precedence properties that can be expressed as link constraints, which allow the comparison of attributes of routers incident on that link. We call these "one-hop precedence properties" since these precedence relations only include comparisons between neighboring routers. For example, the property $\mathbf{G_l}\,(\mathbf{P}a \rightarrow \mathbf{X}b)$ means that all links that originate from a router with attribute a must end in a router with property b. The precedence property mentioned above can be expressed as the global link property $\mathbf{G_l}\,(\bigwedge_i \mathbf{P}p_i \rightarrow \vee_{j\geq i}\mathbf{X}p_j)$.

Given this specification, we can remove all edges from the graph that violate the attribute ordering. Consider an edge (s_1, s_2). If $i > j$, and $p_i \in L(s_1), p_j \in L(s_2)$, then we remove the edge (s_1, s_2) from the graph. Hence no path in the resulting graph can violate the precedence specified by the user. Moreover, any valid path that satisfies the precedence property in the original graph also exists in the resulting graph, and these paths are exactly those in the original graph that satisfy the precedence property.

Note that the user can specify global and one-hop precedence properties simultaneously. These properties on the graph described above are commutative since they involve removing individual links and/or nodes. Given global, and/or one-hop precedence requirements specified by the user, we combine the resulting graph with the trust model described next to find paths of highest "confidence."

4.6.4 Adding variables

It is clear from the preceding examples that expressing precedence properties can be cumbersome. Consider the attribute-value pairs $\langle SecurityLevel, 5\rangle$ and $\langle SecurityLevel, 4\rangle$. Instead of treating this as two separate attributes, it help to treat "Security Level" as a variable that takes on different values in different states (or nodes). When these values are real numbers, we allow users to treat attributes as variables and specify arithmetic operations on these variables. Let S be the security level of a router. The precedence example in Section 4.6.3 can be written as $\mathbf{G_l}\,(\mathbf{P}S \leq \mathbf{X}S)$. This simply states that for every link (u, v), the security level of v must be at least the security level of u. Allowing comparisons of attributes greatly simplifies the specification of precedence relations and global properties such as $\mathbf{G_r}\, 4 \leq S \leq 6$ ("only visit routers with security levels 4, 5, or 6"). We also use the notation $S(s_i)$ to denote the value of variable S in state s_i.

We discuss the use of arithmetic comparisons of attributes in more detail in Section 4.7, where we allow comparisons of trust values for properties between routers.

We now summarize our network model:

$$
\begin{array}{rcl}
AP & : & \text{set of atomic propositions} \\
\text{VAR} & : & \text{set of variables} \\
M & = & \langle S, S_0, R, L, \sigma \rangle \\
S & : & \text{set of routers} \\
S_0 & : & \text{source router} \\
R & \subseteq & S \times S : \text{set of links} \\
L & : & S \rightarrow \mathcal{P}(AP) \\
L & : & R \rightarrow \mathcal{P}(AP) \\
\sigma & : & S \times \text{VAR} \rightarrow \mathbb{R} \\
\sigma & : & R \times \text{VAR} \rightarrow \mathbb{R}
\end{array}
$$

The labeling function L maps a router (or link) to the set of atomic propositions that are true for that router (or link). The valuation function σ maps a router (or link) and a variable to the value of that variable at the router (or link). For simplicity, we will refer to the valuation of x at a router s_i as $\sigma_i(x)$.

4.6.5 Policy language

We summarize the policy language for specifying path properties. This is a fragment of LTL with variables, and constraints on these variables. These variables take different values in different states. We refer to the variables as $\text{VAR} = \{x_1, x_2, \ldots\}$ VAR takes real values in $\mathbb{R}$.

We define the *constraint system* $\mathcal{C} = \{\mathbb{R}, R_1, \ldots, R_n\}$, where $\mathbb{R}$ is the domain of real numbers, each R_i is a relation of arity a_i, such that $R_i : \mathbb{R}^{a_i} \rightarrow \{\mathit{True}, \mathit{False}\}$. An atomic $\mathcal{C}$ constraint over a set of finite variables is of the form $R_i(w_1, \ldots, w_{a_i})$, where each w_i is either a variable or a real-valued constant.

Mathematical equalities and inequalities are examples of constraints. For example, $x_1 \sim c$ and $x_1 \sim x_2 \oplus c$ where $\sim \in \{<, >, =\}$, $\oplus \in \{+, -, \times, \div\}$, and $c \in \mathbb{R}$, are valid

$\mathcal{C}$ constraints.

Let $v : \mathrm{VAR} \rightarrow \mathbb{R}$ be a map or valuation of the variables. We also define v to include the identity map over $\mathbb{R}$, in particular $v(r) = r$ for any $r \in \mathbb{R}$. The interpretation of these constraints is as follows.

$v \models R_i(w_1, \ldots, w_{a_i}) \Leftrightarrow R_i(v(w_1), \ldots, v(w_{a_i}))$

We now define a fragment of CLTL($\mathcal{C}$), i.e., CLTL based on constraint system $\mathcal{C}$. We will call this "policy language" L. We distinguish between two kinds of $\mathcal{C}$ constraints: c_l is a constraint defined with respect to links and c_r is defined for routers. c_l and c_r are relations over variable values at a particular link or router, and additionally, variables in c_l may be prefixed with $\mathbf{P}$ or $\mathbf{X}$ to refer to the values of variables at the incident routers. We will define the semantics of such constraints after presenting the grammar for L:

$$\begin{array}{rcl} a & \in & AP \\ P & ::= & \mathbf{G_r}\,\Phi \mid \mathbf{G_l}\,\Psi \mid \mathbf{G_r}\,\Phi \wedge \mathbf{G_l}\,\Psi \\ \Phi & ::= & c_r \mid a \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \neg\Phi \\ \Psi & ::= & c_l \mid a \mid \mathbf{P}a \mid \mathbf{X}a \mid \Psi \vee \Psi \mid \Psi \wedge \Psi \mid \neg\Psi \end{array}$$

Let $\pi = \langle s_1, \ldots, s_n \rangle$ be a path in the labeled state-transition diagram M. This path represents both routers *and* links. In particular $s_1, s_3, \ldots, s_n$ are routers, and $s_2, s_4, \ldots, s_{n-1}$ are the links $(s_1, s_3), (s_3, s_5), \ldots, (s_{n-2}, s_n)$. Let $\sigma = \sigma_1, \ldots, \sigma_n$ be the sequence of valuations of variables corresponding to routers and links $s_1, \ldots, s_n$ in π. Let S_r be the set of routers in π and S_l be the set of links in π. A discretionary demand by a user includes the policy P and the source and destination pair s, t. The path must be an s, t-path that satisfies P. We define the satisfiability relation for a policy P and path π in labeled state-transition diagram M inductively as follows.

$$
\begin{aligned}
M, \pi &\models \mathbf{G_r}\, \Phi \wedge \mathbf{G_l}\, \Psi \Leftrightarrow M, \pi \models \mathbf{G_r}\, \Phi \wedge M, \pi \models \mathbf{G_l}\, \Psi \\
M, \pi &\models \mathbf{G_r}\, \Phi \Leftrightarrow \forall s_i \in S_r, M, \pi_i \models \Phi \\
M, \pi &\models \mathbf{G_l}\, \Psi \Leftrightarrow \forall s_i \in S_l, M, \pi_i \models \Psi \\
M, \pi_i &\models \neg\Phi \Leftrightarrow M, \pi_i \not\models \Phi \\
M, \pi_i &\models \Phi_1 \vee \Phi_2 \Leftrightarrow M, \pi_i \models \Phi_1 \vee M, \pi_i \models \Phi_2 \\
M, \pi_i &\models \Phi_1 \wedge \Phi_2 \Leftrightarrow M, \pi_i \models \Phi_1 \wedge M, \pi_i \models \Phi_2 \\
M, \pi_i &\models a \Leftrightarrow a \in L(s_i) \\
M, \pi_i &\models \mathbf{P}a \Leftrightarrow a \in L(s_{i-1}) \\
M, \pi_i &\models \mathbf{X}a \Leftrightarrow a \in L(s_{i+1}) \\
M, \pi_i &\models c_r \Leftrightarrow \sigma_i \models c_r \\
M, \pi_i &\models c_l \Leftrightarrow [\ldots, w_j \leftarrow \sigma_i(w_j), \mathbf{P}w_k \leftarrow \sigma_{i-1}(w_k), \mathbf{X}w_l \leftarrow \sigma_{i+1}(w_l), \ldots] \models c_l
\end{aligned}
$$

In other words, policies can specify router policies $\mathbf{G_r}\, \Phi$, link policies $\mathbf{G_l}\, \Psi$, or both. Router policies are boolean combinations of atomic propositions and constraints that must be true at each individual router along a path. Link policies are boolean combinations of atomic propositions, and constraints that must be true at each individual link along a path. Constraints on links can include variables from the incident routers, allowing the user to specify "one-hop precedence properties." We now give some simple examples of our policy language.

Let x represent the number of DoS attacks a node has suffered in the past 24 hours. Let y be the width of cable shielding in mm for links in the network. Let z represent the security level. Let v be the number of virus attacks in the past 24 hours. The policy $\mathbf{G_r}\,(x < 10 \wedge v \leq x) \bigwedge \mathbf{G_l}\,(y > 5 \wedge \mathbf{P}z \leq \mathbf{X}z)$ states that the path must contain routers with at most 10 DoS attacks and the number of virus attacks don't exceed the DoS attacks, links with at least 5mm shielding, and the order of security levels of routers must be non-decreasing.

4.6.6 Graph transformation

Each atomic constraint (node or precedence constraint) can be evaluated to true or false for a router or link as the case may be. Hence these constraints are atomic constraints that evaluate to *true* or *false* and can be used in boolean expressions with other atomic propositions. Each router of link policy p specified as $\mathbf{G_r}\, p$ or $\mathbf{G_l}\, p$

evaluates to *true* or *false* for the node in question. Routers and links, for which the policies evaluate to false, are removed from the graph. Any path from node s to t in this modified graph will satisfy the overall path policy. In the following section, we will discuss the use of shortest path algorithms to accommodate various quantitative trust models. Our choice of policy language is deliberate, since path automata (such as regular expressions) are not compatible with shortest path algorithms for QoP. This stems from the fact that unlike in QoS, where visiting a node twice incurs the latency cost twice, this is not true for QoP. Visiting a node with 5 virus attacks twice only increases the total virus attacks for the path by 5 only once. Since QoP is computed over *sets* of nodes visited, policy languages involving cross-product automata for satisfiability are not compatible with shortest path algorithms, which may yield paths visiting the same node twice. Our policy language makes transformations on the graph, resulting in subgraph of the network, which can be used with shortest path algorithms for computing QoP, without such problems.

4.7 Trust model

Once a user transforms the graph (as described above) of the network satisfying the attribute requirements, the user would like to set up a route to a destination. A naïve solution would be to obtain the shortest path (in terms of hops) to the destination. However, if the network is under attack, some paths are more trustworthy than others. For example, it may be known that there are intruders in the system with physical access to machines. One would like to degrade trust in routers that have lower physical security. It might be known that certain machines have been compromised without knowing the specific machines. In such a case, users may degrade trust for machines run by certain administrators, or for those machines that are running out of date software. Furthermore, certain attributes of routers may be more trustworthy than others. The user may be confident that a router is in a particular domain, but may not be that confident about the router's physical security after a possible break in.

Our next result integrates this notion of threat into the graph-based formalism we proposed so far. We propose a quantitative measure of this interplay between threat and trust as the *confidence* a user has in a router. Users can assign confidence levels to attributes of routers. After specifying the qualitative route properties, users can now choose to optimize their routes based on one or more of their attributes of interest.

As defined below, each attribute a for a router r (or link (u, v)) is augmented with a confidence value $c_r(a)$ ($c_{u,v}(a)$), which can be integer or real valued. We explore the various semantic interpretations of this confidence value and how overall "path-confidence" can be calculated.

Definition 6. *Given a router $s \in S$ with attributes $L(s)$, a user's* **confidence function** *$C : S \times AP \rightarrow \mathcal{R}$ returns the* **confidence level** *for an attribute at a router. We abbreviate the confidence level $C(s, a)$ of a at router s as $c_s(a)$. Similarly we expand the definition to include confidence levels of links. The function $C : S \times S \times AP \rightarrow \mathcal{R}$ returns the* **confidence level** *for an attribute at a router. We abbreviate the confidence level $C(u, v, a)$ of a at link (u, v) as $c_{u,v}(a)$.*

The exact nature of this confidence function will depend on the nature of attributes and how these levels can be composed to compute the confidence value of a path, by combining confidence values of different routers in the path meaningfully. For example, confidence values can represent the probability with which the attribute-value pair is true or not. It can also represent the number of incidents reported by an intrusion detection system or positive reports submitted by users. In each case these confidence levels must be combined meaningfully to reflect overall path confidence, which we describe in the next section. We discuss how we can compute paths of high overall confidence based on confidence levels of attributes of routers along the path.

4.7.1 Trusted paths

We refer to any simple (no repeated vertices) path from router a to router b as an a, b-path. Similarly an a, b-walk is a path from a to b that may repeat vertices and edges. We assume that the user/sender is connected through access point a, and that the destination is either b or a user whose access point is b. In either case we treat a and b as the endpoints of communication.

We now define the path-confidence of an attribute.

Definition 7. *The* **path confidence** *$\mathcal{C}_\pi(a)$ for an attribute a along an u, v-path π is obtained by applying a* **combiner function** *$K(c_1(a), \ldots, c_n(a))$ that takes all the confidence levels $c_i(a)$ of the n nodes s_i along the path π from u to v ($s_1 = u, s_n = v$), and returns a single confidence value for the path in $\mathcal{R}$.*

We assume that a combiner function is applied with respect to a single attribute, and omit the "(a)" part in $\mathcal{C}_\pi(a)$ and $c_i(a)$ above for clarity. Users may also want to optimize over the values of variables. For example, let D be the number of DoS attacks a router has suffered within a certain time window. The user may want to find a path that minimizes the sum total of D along a path.

Definition 8. *The* **path confidence** $\mathcal{C}_\pi(D)$ *for an attribute variable D along an u, v-path π is obtained by applying a* **combiner function** $K(D(s_1), \ldots, D(s_n))$ *that takes all the variable values $D(s_i)$ of the n nodes s_i along the path π from u to v ($s_1 = u, s_n = v$), and returns a single confidence value for the path in $\mathcal{R}$.*

In Section 4.8 we discuss how a user can optimize path-confidence for multiple attributes (multiple combiners). We also assume that if confidence levels are also associated with link attributes, links are subdivided to include a node that represents the link. Hence all confidence levels will be associated with nodes in the graph, which allows us to use shortest path algorithms in a consistent manner.

We now explore different combiner functions and how they apply to different models of trust. To illustrate, consider the concept of "weakest link." There may be routers that are highly vulnerable, and it is extremely likely that they will be chosen for attack. The path confidence in this case can be defined as the *minimum* of all confidence values of routers along the path. Here $K(c_1, \ldots, c_n) = \min\{c_1, \ldots, c_n\}$. So when a user needs to pick a path based on its combined confidence value, he or she can avoid paths with low path confidence.

Also consider the following example. A user may conclude that the DoS vulnerability of a router is proportional to the number of incoming links. Hence the user would like a path that minimizes the average sum of incoming links over all routers along a path, but also does not include any nodes with very high connectivity. The user can first eliminate routers with incoming links beyond a certain threshold and then minimize the average. In this case the user can also use a second order statistic such as variance to decide which path has the best "Quality of Protection" for the given scenario.

First we focus on the multiplicative combiner. A multiplicative measure of path confidence can be used to model various properties of interest to a user: high probability of success of delivery, high probability of no information leakage, high probability that routers along a path will not collude, etc. This model assumes that events at each router are independent and their success probabilities can therefore be multiplied to calculate overall success probability for the path. In the next subsection, we explore this in some detail and describe efficient algorithms to compute path confidence values using a multiplicative combiner function.

4.7.2 Multiplicative combiners

We now present our results for multiplicative combiners. We consider the case when $K(c_1, \ldots, c_n) = c_1 \ldots c_n$, the product of confidence levels of nodes along a path. This multiplicative model of path confidence we focus on in this subsection, applies to confidence levels that were computed independently along a path. In this model, a user assigns confidence levels based on the probability of "good things happening" at each node. Assuming independence, the probability of the desired property being true along the entire path is simply the product of all the confidence levels. We now present an efficient method for computing paths of high path confidence under the multiplicative model.

The main idea behind computing paths of high confidence is that by applying the correct weights to edges in a network connectivity graph, we can use shortest path algorithms (that use additive weights) to find paths with highest overall confidence (based on multiplicative weights).

Consider the directed graph G that represents the connectivity of routers specified by the labeled state-transition diagram M. As mentioned earlier, links with confidence levels can be subdivided to include a node that represents that link. For each $s \in S$, we now assign $-\ln(c_s)$ to be the *weight* of all incoming edges to s, i.e., $\{(u,s) \in R : u \in S\}$. Note that all weights are non-negative since confidence levels are in the range $[0,1]$. We now have a weighted directed graph G. Consider a source node a and a destination node b.

Lemma 1. *Let s be the sum of weights on the a,b-path π in G. The path confidence $\mathcal{C}_\pi$ of π is equal to e^{-s}.*

Proof. Let $c_1, \dots, c_n$ be the confidence levels of all the nodes in π except a. $\mathcal{C}_\pi = c_1 c_2 \dots c_n$ since $c_a = 1$. Now $s = \sum_{i=1}^{n} -\ln(c_i) = -\sum_{i=1}^{n} \ln(c_i) = -\ln(c_1 c_2 \dots c_n)$. Hence $e^{-s} = e^{\ln(c_1 c_2 \dots c_n)} = c_1 c_2 \dots c_n = \mathcal{C}_\pi$.

Note that if for there exists a $c_i = 0$, then the path confidence is 0. Moreover, $s = \infty$ since $-\ln(0) = \infty$ and $e^{-s} = 0$, so there is no discrepancy for confidence levels of 0. Essentially, any path which includes a node of 0 confidence will not be chosen by the user. □

Lemma 2. *For any two a,b-paths π_1, π_2 with total weights w_1, w_2, we have $w_1 \leq w_2$ if and only if $\mathcal{C}_{\pi_1} \geq \mathcal{C}_{\pi_2}$.*

Proof. From Lemma 1 we have that $w_1 \leq w_2 \Leftrightarrow -w_1 \geq -w_2 \Leftrightarrow e^{-w_1} \geq e^{-w_2} \Leftrightarrow \mathcal{C}_{\pi_1} \geq \mathcal{C}_{\pi_2}$. □

Theorem 1. *The k-shortest a,b-paths in G correspond to the k a,b-paths of highest path confidence in G.*

Proof. This follows from Lemma 2 since if we order all the a,b-paths in G in increasing order of weight, they are ordered in decreasing order of path confidence.

□

Since all edge weights are non-negative, Theorem 1 allows us to apply k-shortest simple (loopless) path algorithms to find cycle-free paths of highest confidence. For example, Dijkstra's algorithm is the special case when $k = 1$ and will yield a path with maximum path confidence. Several algorithms have been proposed for obtaining the k shortest simple paths in a directed graph. The best known worst case time complexity of these algorithms is $O(kn(m + n \log n))$ [Yen71; Yen72]. Hershberger et al. [HMS03] propose an algorithm that provides a $\Theta(n)$ improvement in most cases. For small k (for example, the user may want the 3 highest confidence paths) these algorithms are efficient for all practical purposes. Hershberger et al. [HMS03] provide results of their algorithm for large graphs (e.g., 5000 nodes, 12000 edges) based on real GIS (Geographic Information Services) data for road networks in the United States.

In addition to the models we present in this section, we argue that the ability to specify both threat and trust relationships using a combined metric is extremely powerful. We plan to study how these values can vary over time, using sensitivity analysis, stochastic analysis and other techniques. In the next section, we present three example scenarios that showcase the benefits of our new framework.

4.7.3 Additive combiners

We consider the case when $K(c_1, \ldots, c_n) = c_1 + \ldots + c_n$, the sum of confidence levels of nodes along a path. Finding simple paths of highest path-confidence is NP-hard. For example, the Hamiltonian Path problem can be reduced to finding the longest path in a graph with uniform edge-weights, and then verifying whether it is a Hamiltonian path or not.

Hence we look at the problem, where we attempt to minimize the confidence values. For better intuition, we refer to these as diffidence values, and correspondingly refer to path-confidence as path-diffidence. Paths of least diffidence can be solved trivially by using shortest path algorithms. This model can be used in cases where intrusion detection systems may produce negative reports for nodes. Users may want to find paths that minimize the sum of negative reports along the path, corresponding to a path with the least number of known problems. Similarly, a node's incoming degree can be a measure of vulnerability to DoS. A user may want to find a path with the least number of total incoming edges, which could imply a lower amount of vulnerability of the path to DoS.

A user may also want to find a path where the average confidence (or average diffidence) for each node along a path is minimized (resp. maximized). We address the problem of average combiners below, and show that finding solutions for this demand is NP-hard.

4.7.4 Weakest link

As mentioned earlier, the confidence of the path is the minimum confidence level of nodes in the path, $K(c_1, \ldots, c_n) = \min\{c_1, \ldots, c_n\}$. The path of highest confidence can be computed by sorting the links based on weight. First all links are removed from the graph, and links are added back iteratively in descending order of weight. At each iteration, if a path from s to t exists, then it will be the path of highest confidence. The same can be done for computing paths of least diffidence, where the confidence level is the maximum of confidence level of routers along the path.

For example, consider an attribute that measures DoS resilience. Furthermore the user is certain that there is a DoS attack in the network and would like a path with the highest DoS resilience. Since the DoS resilience of a path is only as good as its weakest link, the user can use this combiner to find a path of highest confidence, or DoS resilience.

4.7.5 Average combiners

We consider the case when the confidence or diffidence $K(c_1, \ldots, c_n) = \frac{c_1 + \ldots + c_n}{n}$, and the user desires a path of least average cost (or least diffidence) or highest average cost (highest confidence).

For example, the user may desire a path that minimizes the average incoming degree for each node. Singh et al. [SCRD04] describe an eclipse attack in overlay networks where malicious nodes are identified by having a higher in-degree. In an eclipse attack, a group of malicious nodes attempts to corrupt routing tables of other nodes in the network, such that all communication in the network is directed through malicious nodes. The authors observe that malicious nodes in this setting would have a high incoming degree and propose an auditing mechanism to ascertain the incoming degree of nodes. In particular, malicious nodes cannot hide their incoming degree because of their proposed anonymous auditing mechanism. A reasonable demand would be to find a path with the lowest minimum average degree, improving the overall confidence in the path with respect to the eclipse attack. This can be done after eliminating nodes above a certain threshold of incoming degree to avoid the obviously malicious nodes.

We show that these problems are NP-hard by reducing the s, t-Hamiltonian Path problem to finding the minimum or maximum average cost path.

Definition 9. Hamiltonian Path Problem (HP): *Given a directed graph $G = (V, E)$ find an s, t-path that visits all vertices in V. Such a path is called a* **Hamiltonian path***.*

HP is NP-complete.

Definition 10. **s, t-Hamiltonian Path Problem (s, t-HP):** *Given a directed graph $G = (V, E)$ and vertices $s, t \in V$, find an s, t-path that visits all vertices in V. Such a path is called an* **s, t-Hamiltonian path**

s, t-HP is NP-complete. It is easy to show this by reducing HP to s, t-HP. Given a graph G, construct G' by adding vertices s, t, and the directed edges (s, v) and (v, t) for all vertices $v \in V$. G' has an s, t-Hamiltonian path if and only if G has a Hamiltonian path. Hence s, t-HP is NP-complete.

Definition 11. Minimum Average Cost Simple-Path Problem (MinACSPP)*: Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-path p that minimizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of p divided by the number of vertices in p.*

Maximum Average Cost Simple-Path Problem (MaxACSPP)*: Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-path p that maximizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of p divided by the number of vertices in p.*

Theorem 2. *The Minimum Average Cost Simple-Path Problem (MinACSPP) is NP-hard.*

Proof. We reduce the s, t-HP to MinACSPP. Given a graph $G = (V, E)$, and vertices $s, t \in V$, assign the weight 1 to all vertices except t. Assign the weight $1+\delta$ to t. Any s, t-path of length (number of vertices) n will have average cost $\frac{(n-1)+(1+\delta)}{n} = 1 + \frac{\delta}{n}$. This average cost is minimized for largest possible $n = |V|$. Hence the solution to MinACSPP will yield an s, t-path that visits $|V|$ vertices if and only if an s, t-Hamiltonian path exists in G. Hence MinACSPP is NP-hard. □

Theorem 3. *The Maximum Average Cost Simple-Path Problem (MaxACSPP) is NP-hard.*

Proof. We reduce the s, t-HP to MinACSPP. Given a graph $G = (V, E)$, and vertices $s, t \in V$, assign the weight 1 to all vertices except t. Assign the weight $1 - \delta$ to t (where $\delta < 1$). Any s, t-path of length (number of vertices) n will have average cost $\frac{(n-1)+(1-\delta)}{n} = 1 - \frac{\delta}{n}$. This average cost is maximized for largest possible $n = |V|$. Hence the solution to MaxACSPP will yield an s, t-path that visits $|V|$ vertices if and only if an s, t-Hamiltonian path exists in G. Hence MaxACSPP is NP-hard. □

These results imply that in general it is very hard to compute paths that minimize/maximize path diffidence/confidence. A natural question to ask is whether the shortest average path for various restrictions on hop-length can be computed. In Section 4.8.3 we show that these related problems are also NP-hard.

4.7.6 Minimum variance

We consider the case when the diffidence $K(c_1, \ldots, c_n) = \frac{c_1^2+\ldots+c_n^2}{n} - (\frac{c_1+\ldots+c_n}{n})^2$, and the user desires a path of least variance or least diffidence.

For example, the user may want to pick a path with the most consistent (as measured by low variance) set of confidence values within an acceptable range.

We show that the problem of minimizing variance is NP-hard by reducing s, t-HP to finding the minimum variance path.

Definition 12. Minimum Variance Simple-Path Problem (MVSPP)*: Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$ such that $(s, t)\neg \in E$, find an s, t-path p that minimizes the variance of weights for the set of vertices in p.*

We assume that s, t are not directly connected by a single edge, because then the solution is trivial. The path $\langle s, t\rangle$ has variance 0 and is the minimum variance path.

Theorem 4. *The Minimum Variance Simple-Path Problem is NP-hard.*

Proof. We reduce the s, t-HP to MVSPP. Given a graph $G = (V, E)$, and vertices $s, t \in G$, assign the weight 1 to all vertices other than t. Assign the weight $1 + \delta$ to t.

Any s, t-path of length (number of vertices) n will have variance

$$= \frac{(n-1)1^2 + (1+\delta)^2}{n} - \frac{(n+\delta)^2}{n^2} \tag{4.1}$$

$$= \frac{n+\delta^2+2\delta}{n} - \frac{n^2+\delta^2+2n\delta}{n^2} \tag{4.2}$$

$$= \frac{n^2+n\delta^2+2n\delta}{n^2} - \frac{n^2+\delta^2+2n\delta}{n^2} \tag{4.3}$$

$$= \frac{\delta^2(n-1)}{n^2} \tag{4.4}$$

For any fixed δ, since $n \geq 3$ (by assumption that $(s, t) \notin E$), the variance is minimized for largest possible $n = |V|$. Hence the solution to MVSPP will yield an s, t-path that visits $|V|$ vertices if and only if an s, t-Hamiltonian path exists in G. Hence MVSPP is NP-hard.

□

4.7.7 Approximation

For minimum/maximum average and minimum variance weight s, t-paths picking a large value for δ such as n^2, the approximate solutions to these problems will closely approximate solutions to the longest s, t-path problem. Karger et al. [KMR93] show that unless **P = NP**, there is no polynomial time algorithm that can find a path of length $n - n^\epsilon$ for any $\epsilon > 1$. This indicates that there is very little hope to approximate these problems.

4.7.8 Measurement

Before we continue with our discussion on multiple combiners, an important question is how confidence levels can be measured. We based some of our examples on intrusion detection system (IDS) reports. For example, an IDS can record the number of virus or worm intrusions for systems on the network and share these reports with users, possibly attached to the network graph presented to the user. Gossip protocols could be used to share information about neighboring routers, although care must be taken to ensure the integrity of such approaches. Singh et al. [SCRD04] discuss an approach whereby routers can query other routers anonymously to report on their routing table information, and show how the node being queried cannot falsify responses. Various numerical attributes (using attribute variables) can be certified (e.g., security level) and obtained from the router itself, or through a lookup service. When accessing lookup services, it is important to know that queries can leak the policies of users. Hence users must request properties of several (or all) routers to keep their policy contents secret. The use of gossip protocols [CRB01] allows users to passively collect information about other routers, and is a good solution for certified attributes. In this thesis we focus on the issue of *using* confidence values by providing a general model that allows the use of several confidence assessment techniques.

4.8 Multiple combiners

Consider the case where a user may want to maximize confidence along a path for two or more attributes. In our model this is the same as applying two or more combiners to the labeled state-transition diagram.

There has been considerable work in the QoS community that addresses finding network paths that minimize multiple constraints. It can be shown that minimizing two additive or multiplicative (or a combination of the two) constraints is NP-hard. Specifically, given n attributes with the additive combiner, and thresholds $L_1, \ldots, L_n$ for each attribute, finding a path with costs $c_1, \ldots, c_n$ for each attribute such that $c_1 \leq L_1, \ldots, c_n \leq L_n$ is NP-complete [WC96]. This immediately implies hardness results for multiple attributes in the additive or multiplicative models.

4.8.1 Unifying multiple attributes

We present an approach that unifies confidence for each node, after which a single combiner can be applied. Specifically, multiple attributes are treated as a single attribute with a unified confidence value, after which our results for single attributes can be applied.

Tractable additive models

Consider the two attributes "DoS attacks" and "Worm attacks." Each attribute has diffidence equal to the number of intrusion detection reports for that attack. A user may want to pick a path that minimizes the number of "DoS and Worm attacks." In this case the single attribute "DoS and Worm attacks" can be unified by adding their diffidence values. The user can also choose to weight each attribute. For example, the user may consider DoS attacks more important, and unify the DoS Attack and Worm Attack diffidences d and w as $0.8d + 0.2w$. In fact this is semantically the same as consider each combiner separately, and minimizing a linear combination of each additive combiner.

Formally, consider the n attribute variables $a_1, \ldots, a_n$. For a given path π, their individual path confidences are represented as $\mathcal{C}_\pi(a_1), \ldots, \mathcal{C}_\pi(a_n)$. As mentioned earlier, finding a path π such that $\mathcal{C}_\pi(a_1) \leq L_1, \ldots, \mathcal{C}_\pi(a_n) \leq L_n$ for supplied thresholds $L_1, \ldots, L_n$ is NP-complete. However, finding a path that minimizes $w_1\mathcal{C}_\pi(a_1) + \ldots + w_n\mathcal{C}_\pi(a_n)$ is equivalent to minimizing the additive cost $\mathcal{C}_\pi(a)$ of unified attribute a, where for each node k, $c_k(a) = w_1c_1 + \ldots + w_nc_n$. For attribute variables, we would have $a(s_k) = a_1(s_k) + \ldots + a_n(s_k)$. This is easy to prove as in [Jaf84].

This model would apply more easily to attributes with low correlation. In general it is reasonable to assume that DoS and Worm attacks are unrelated. For example, the DoS vulnerability of the node is a function of its connectivity, whereas vulnerability to worms is related to the current version of software. While unifying attributes with a high degree of correlation can be done by more complicated unifiers, we use the linear combination model for its tractability.

Tractable multiplicative models

We focus on the model where confidence levels are equal to the probability that the attribute is true. We assume that these probabilities are independent. In this case, multiple attributes at a router can be unified into a single attribute using boolean connectives. It is easy to compute the overall probability for expressions such as $a_1 \wedge \ldots \wedge a_n$ or $a_1 \vee \ldots \vee a_n$ using standard combinatorial rules. We assume unifiers of this form. For example, unified attribute a defined as $a_1 \wedge a_2$ will have the confidence $c(a_1)c(a_2)$, while a defined as $a_1 \vee a_2$ will have the confidence $c(a_1)+c(a_2)-c(a_1)c(a_2)$. This allows us to meaningfully unify attributes under the independent probability model. Unified attributes now have a single probability (or confidence) at individual nodes, and the multiplicative combiner can be use to find paths of highest confidence.

For a more complicated combination of boolean connectives we assume a user supplied formula for calculating the overall probability or the use of available tools. Composing events as arbitrary boolean expressions is common in Fault Tree Analysis (FTA) [FTA81] and several FTA tools such as Galileo [CS00] and SAPHIRE [sap] are available for computing the overall probability of the defined event.

We now address more complicated privacy demands where a user may want to visit k or more distinct nodes to make a traceback attack harder to execute by an attacker. In other words, an attacker trying to expose the location of the user will have to retrace the path through these routers, and a user would like to select such a path with high confidence.

4.8.2 Visit k distinct nodes

Consider an attribute variable such as node ID d. We consider the case when $K(d(s_1), \ldots, d(s_n)) = 1$ if $n = k$ and 0 otherwise. Finding an s, t-path of highest non-zero confidence is equivalent to finding a simple path from s to t that visits exactly k or at least k nodes (s, t-k-path or s, t-k^+-path). Such a property would be of interest to thwart traceback attacks to expose a user's location, but this problem is NP-complete [AYZ95]. Indeed if we set $k = |V|$, then a solution to this problem will yield an s, t-Hamiltonian path for any graph $G = (V, E)$ if and only if one exists. Hence the minimum weight simple k-path and k^+-path problems are NP-hard optimization problems. We call these problems k-MWSP and k^+-MWSP.

Note that the problem of finding an s, t-path with at most k nodes (s, t-k^{-}-path) is easily solved by finding the shortest path from s to t and testing whether the length is at most k, however we are interested in k as a lower bound for security against traceback attacks.

If we relax the restriction on simple paths to allow walks (vertices and edges can be repeated), the problem of finding an s, t-k-walk is trivially solvable for strongly connected graphs. Since the graph is strongly connected, a walk can be constructed that will visit k distinct nodes by first finding the shortest path from s to t. If there are more than k nodes in this path, the desired walk does not exist. If there are less than k nodes in this walk, then neighbors to this walk can be successively inserted to increase the distinct nodes visited by 1 with each iteration. More precisely, at the beginning of each iteration, there will exist at least one vertex v in the walk with a neighbor w in the list of unvisited vertices. This is guaranteed by the fact that the graph is strongly connected. Replace v in the walk with v, w, v.

We now present a high level algorithm for finding an s, t-walk in any directed graph that visits at least k vertices. For a directed graph G, this algorithm finds an s, t-walk that visits the most possible distinct vertices, and hence will trivially satisfy the "at least k" requirement. If the path returned by this algorithm visits fewer than k distinct vertices, then we know that no such path exists.

```
Decompose graph into Strongly Connected Components (SCC)
//Linear time decomposition $O(|V| + |E|)$ [Tar72]
for each SCC $S$ do
   assign weight $-|S|$ to each incoming edge to $S$
   //where $|S|$ is the number of vertices in $S$
end for
let $S_s, S_t$ be the components containing $s, t$ respectively
find the minimum cost path from $S_s$ to $S_t$
//The SCC graph is a DAG, and minimum cost algorithms for graphs with negative weights can be applied
```

This algorithm yields a path p from S_s to S_t in the SCC graph that maximizes the sum of vertices of each SCC (or cost $-c$). Starting from s, a walk can be constructed that visits all nodes in each SCC of p, and ending at t. This will be an s, t-walk that visits the maximum number of distinct vertices (c vertices). If $c \geq k$ we can use this walk as an s, t-k^{+}-walk. If $c < k$, no such walk exists.

For directed graphs in general, we are not aware of the complexity of finding a walk that visits exactly k distinct vertices. However, we show that the minimum cost version of finding a walk that visits k distinct vertices, and optimizes another confidence metric is NP-hard. Hence in general, it is hard to find optimal walks or paths in networks with a specified number of distinct nodes even if we allow repetition of nodes.

Definition 13. k-Distinct Vertex Minimum Weight Walk Problem (k-MWWP): *Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-walk p that visits $k < |V|$ distinct vertices (we will call this an s, t-k-walk), and minimizes the weight of walk p. The weight $w(p)$ of walk p is defined as the the total additive cost of the* set *of vertices in p. $w(p) = \sum_{v \in p} w(v)$. Hence the cost of visiting a vertex is incurred only once.*

The Vertex Weighted k-Minimum Tree Problem is NP-hard [FHJM94]. We reduce this to s, t-k-VMT, and in turn to k-MWWP to prove NP-hardness of k-MWWP.

Definition 14. Vertex Weighted k-Minimum Tree Problem (k-VMT): *Given an undirected graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, find a tree T in G with $k < |V|$ vertices (we call this a k-tree[2]), where T is of minimum weight. The weight $w(T)$ of T is the sum of weights of the set of vertices in T. $w(T) = \sum_{v \in T} w(v)$.*

Definition 15. Vertex Weighted s, t-k-Minimum Tree Problem (s, t-k-VMT): *Given an undirected graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, find a tree T in G with $k < |V|$ vertices containing specified vertices s and t. (we call this an s, t-k-tree), where T is of minimum weight. The weight $w(T)$ of T is the sum of weights of the set of vertices in T. $w(T) = \sum_{v \in T} w(v)$.*

Lemma 3. *Vertex Weighted s, t-k-Minimum Tree Problem (s, t-k-VMT) is NP-hard.*

Proof. We reduce k-VMT to s, t-k-VMT.

Given k and an undirected graph $G = (V, E)$ with vertex weights $w(v)$, construct $G' = (V', E')$ in the following way. Assume some ordering $v_1, \ldots, v_n$ of vertices in V. Start with a copy of G, and for each vertex $v_i \in V$ add two new vertices s_i and t_i. Add the edges (v_i, s_i) and (v_i, t_i). Let $M = \sum_1^n w(v_i)$. Assign the weight $M + 1$ to s_i and t_i. In addition, add the vertices s and t and the edges (s, s_i) and (t, t_i) for all $i = 1, \ldots, n$. Assign weights $\delta = 1$ to s, t. This new graph contains $3n + 2$ vertices, and $|E| + 4|V|$ edges. G', along with s, t and $k + 4$ is used as input to the s, t-k-VMT problem. We show that G has a k-tree of cost $\leq c$ if and only if G' has an s, t-$(k + 4)$-tree of cost $\leq c + 2(M + 1) + 2\delta$, where $c \leq M$.

Clearly if G has a k-tree T of cost $\leq c$, then we we can add vertices s_i, t_i for some $v_i \in T$, along with s, t and edges (v_i, s_i), (v_i, t_i), (s, s_i), (t, t_i) to obtain an s, t-$(k+4)$-tree T' in G', where $w(T') \leq c + 2(M + 1) + 2\delta$.

Likewise, let T' be an s, t-$(k + 4)$-tree in G' with weight $\leq c + 2(M + 1) + 2\delta$, where $c \leq M$. We argue that T' contains k vertices from V, and hence includes only two

[2] Fischetti et al. [FHJM94] define a k-tree to have k edges, however trees with k edges have $k + 1$ vertices, and hence our definition is equivalent

vertices s_i and t_j for some particular values of i, j. Since $s, t \in V(T')$, we know that must be at least two such vertices. Consider the case when there are more than two such vertices. We have $w(T') \geq 3(M+1) + 2\delta$. But since $w(T') \leq c + 2(M+1) + 2\delta$, and $c \leq M$, we have $w(T') < 3(M+1) + 2\delta$, which is a contradiction. Hence there are only two vertices of weight $M+1$. Let T be the k-vertex embedding of T' in G. We know that T is a tree because any two vertices in T are connected in T', but cannot be connected through s, t, s_i, s_j. T is therefore connected and is a tree. Furthermore $w(T) = w(T') - 2(M+1) - 2\delta \leq c$.

It follows that a minimum s, t-$(k+4)$-tree T' of G' can be transformed into the minimum k-tree of G, and we have that s, t-k-VMT is NP-hard.

□

Theorem 5. *k-Distinct Vertex Minimum Weight Walk Problem (k-MWWP) is NP-hard.*

Proof. We reduce s, t-k-VMT, which is NP-hard from Lemma 3 to k-MWWP.

Given an undirected graph $G = (V, E)$ with vertex weights $w(v)$, k, and $s, t \in V$, create the directed graph $G' = (V' = V, E')$, where each undirected edge $(u, v) \in E$ is replaced by directed edges (u, v) and (v, u) in E'.

We claim that G contains an s, t-k-tree of weight $\leq c$ if and only if G' contains an s, t-k-walk of weight $\leq c$.

If T is an s, t-k-tree of weight $\leq c$. Consider the embedding T' of T in G', where each undirected edge (u, v) in T is replaced with the corresponding directed edges (u, v) and (v, u) in T'. T' is a strongly connected subgraph of G' with k distinct vertices. Hence we can construct a walk p from s to t using only vertices and edges in T', yielding an s, t-k-walk of the same weight, which is $\leq c$.

Let p be an s, t-k-walk of G' of cost $\leq c$. Consider the embedding G_p of p in G, where directed edges of p are replaced by undirected edges in G. G_p is a connected subgraph of G. Let T_p be a spanning tree of G_p (this can be computed in polynomial time). T_p is an s, t-k-tree of the same weight $\leq c$.

It follows that a minimum weight s, t-k-walk in G' can be transformed into a minimum weight s, t-k-tree in G, and hence k-MWWP is NP-hard.

□

Since finding minimum weight s, t-walks of length k is NP-hard, an interesting question is whether one can find minimum weight s, t-walks with *at least* k distinct vertices. Earlier we showed that finding s, t-k^+-walks without a cost metric can be done in polynomial time. However we show that the minimum cost version of finding s, t-k^+-walks (we call this problem k^+-MWWP) is NP-hard.

Lemma 4. *Minimum cost k^+-tree problem is NP-hard (k^+-VMT).*

Proof. We reduce k-VMT to k^+-VMT.

Given a graph G we claim that there exists a k-tree T of cost $\leq c$ if and only if there exists a k^+-tree T' of cost $\leq c$.

Clearly, a k-tree T of cost $\leq c$ is also a k^+-tree of cost $\leq c$.

Now consider a k^+-tree T' of cost $\leq c$. Consider any subtree T of T' with k vertices. T is a k-tree with $w(T) \leq c$.

It follows that the minimum weight k^+-tree of G yields a minimum weight k-tree of G, and hence the k^+-tree problem is NP-hard.

□

Lemma 5. *Vertex Weighted s, t-k-Minimum Tree Problem (s, t-k-VMT) is NP-hard.*

Proof. We reduce k^+-VMT to s, t-k-VMT.

Given k and an undirected graph $G = (V, E)$ with vertex weights $w(v)$, construct $G' = (V', E')$ in the following way. Assume some ordering $v_1, \ldots, v_n$ of vertices in V. Start with a copy of G, and for each vertex $v_i \in V$ add two new vertices s_i and t_i. Add the edges (v_i, s_i) and (v_i, t_i). Let $M = \sum_1^n w(v_i)$. Assign the weight $M + 1$ to s_i and t_i. In addition, add the vertices s and t and the edges (s, s_i) and (t, t_i) for all $i = 1, \ldots, n$. Assign weights $\delta = 1$ to s, t. This new graph contains $3n + 2$ vertices, and $|E| + 4|V|$ edges. G', along with s, t and $k + 4$ is used as input to the s, t-k-VMT problem. We show that G has a k^+-tree of cost $\leq c$ if and only if G' has an s, t-$(k + 4)^+$-tree of cost $\leq c + 2(M + 1) + 2\delta$, where $c \leq M$.

Clearly if G has a k^+-tree T of cost $\leq c$, then we we can add vertices s_i, t_i for some $v_i \in T$, along with s, t and edges (v_i, s_i), (v_i, t_i), (s, s_i), (t, t_i) to obtain an s, t-$(k + 4)^+$-tree T' in G', where $w(T') \leq c + 2(M + 1) + 2\delta$.

Likewise, let T' be an s, t-$(k+4)^+$-tree in G' with weight $\leq c+2(M+1)+2\delta$, where $c \leq M$. We argue that T' contains at least k vertices from V, and includes only two vertices s_i and t_j for some particular values of i, j. Since $s, t \in V(T')$, we know that must be at least two such vertices. Consider the case when there are more than two such vertices. We have $w(T') \geq 3(M+1)+2\delta$. But since $w(T') \leq c+2(M+1)+2\delta$, and $c \leq M$, we have $w(T') < 3(M + 1) + 2\delta$, which is a contradiction. Hence there are only two vertices of weight $M + 1$. Let T be the embedding of T' in G using only the vertices in V. We know that T is a tree because any two vertices in

T are connected in T', but cannot be connected through s, t, s_i, s_j. T is therefore connected and is a k^+-tree. Furthermore $w(T) = w(T') - 2(M+1) - 2\delta \leq c$.

It follows that a minimum s,t-$(k+4)^+$-tree T' of G' can be transformed into the minimum k^+-tree of G, and we have that s,t-k^+-VMT is NP-hard.

□

Theorem 6. *k^+-Distinct Vertex Minimum Weight Walk Problem (k^+-MWWP) is NP-hard.*

Proof. We reduce s,t-k^+-VMT, which is NP-hard from Lemma 3 to k^+-MWWP.

Given an undirected graph $G = (V, E)$ with vertex weights $w(v)$, k, and $s, t \in V$, create the directed graph $G' = (V' = V, E')$, where each undirected edge $(u, v) \in E$ is replaced by directed edges (u, v) and (v, u) in E'.

We claim that G contains an s,t-k^+-tree of weight $\leq c$ if and only if G' contains an s,t-k^+-walk of weight $\leq c$.

If T is an s,t-k^+-tree of weight $\leq c$. Consider the embedding T' of T in G', where each undirected edge (u, v) in T is replaced with the corresponding directed edges (u, v) and (v, u) in T'. T' is a strongly connected subgraph of G' with at least k distinct vertices. Hence we can construct a walk p from s to t using only vertices and edges in T', yielding an s,t-k^+-walk of the same weight, which is $\leq c$.

Let p be an s,t-k^+-walk of G' of cost $\leq c$. Consider the embedding G_p of p in G, where directed edges of p are replaced by undirected edges in G. G_p is a connected subgraph of G. Let T_p be a spanning tree of G_p (this can be computed in polynomial time). T_p is an s,t-k^+-tree of the same weight $\leq c$.

It follows that a minimum weight s,t-k^+-walk in G' can be transformed into a minimum weight s,t-k^+-tree in G, and hence k^+-MWWP is NP-hard.

□

Approximation

While we have shown that finding node-weighted minimum weight s,t-paths in a network that visit k or at least k distinct nodes is NP-hard in general, approximation algorithms may yield acceptable solutions. We leave this to future work, but mention relevant research here. For the simpler node-weighted minimum weight k-cardinality tree problem Mladenović and Urošević [MU04] present a heuristic based on variable neighborhood search and provide performance results under various scenarios. Blum and Ehrgott [BE03] show that problem can be solved in polynomial time if the graph contains "exactly one trough." They also present several local search heuristics for the problem, and provide an extensive discussion on related solutions for this problem. Hence, it may be useful to compute approximate solutions based on these heuristics to find paths of acceptable confidence, if not the highest confidence.

4.8.3 Scoped minimum average cost

We now return to the problem of finding the minimum average cost path in a graph and explore the complexity of s, t-k-walks of minimum average cost.

Definition 16. k **Minimum Average Cost Simple-Path Problem (**k**-MinACSPP):** *Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-k-path p that minimizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of p divided by the number of vertices in p.*

k^+ **Minimum Average Cost Simple-Path Problem (**k^+**-MinACSPP):** *Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-k^+-path p that minimizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of p divided by the number of vertices in p.*

k^- **Minimum Average Cost Simple-Path Problem (**k^-**-MinACSPP):** *Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-k^--path p that minimizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of p divided by the number of vertices in p.*

k **Minimum Average Cost Walk Problem (**k**-MACWP):** *Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-k-walk p that minimizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of the set of vertices in p divided by the cardinality of the set of vertices in p.*

Theorem 7. *k Minimum Average Cost Simple-Path Problem (k-MinACSPP) is NP-hard.*

Proof. We reduce s, t-HP to k-MinACSPP.

Given a graph $G = (V, E)$, and vertices $s, t \in V$, simply specifying $k = |V|$, and $w(v) = 1$ for all $v \in V$, k-MinACSPP will return an s, t-Hamiltonian path in G if and only if it exists.

□

Theorem 8. *k^+ Minimum Average Cost Simple-Path Problem (k^+-MinACSPP) is NP-hard.*

Proof. We reduce s,t-HP to k^+-MinACSPP.

Given a graph $G = (V, E)$, and vertices $s, t \in V$, simply specifying $k = |V|$, and $w(v) = 1$ for all $v \in V$, k^+-MinACSPP will return an s,t-Hamiltonian path in G if and only if it exists.

□

Theorem 9. *k^- Minimum Average Cost Simple-Path Problem (k^--MinACSPP) is NP-hard.*

Proof. We reduce s,t-HP to k^--MinACSPP.

Given a graph $G = (V, E)$, and vertices $s, t \in V$, assign the weight 1 to all vertices other than t. Assign the weight $1+\delta$ to t. Any s,t-path of length n will have average cost $\frac{(n-1)+(1+\delta)}{n} = 1+\frac{\delta}{n}$. This average cost is minimized for largest possible $n = |V|$. Hence for $k = n$, the solution to k^--MinACSPP will yield an s,t-path that visits $|V|$ vertices if and only if an s,t-Hamiltonian path exists in G. Hence k^--MinACSPP is NP-hard.

□

Theorem 10. *k Minimum Average Cost Walk Problem (k-MACWP) is NP-hard.*

Proof. We can reduce k-MWWP to k-MACWP.

Given a graph $G = (V, E)$, vertices $s, t \in V$, and k, we need to find the minimum cost walk from s to t with exactly k vertices.

We use G, s, t and k as input to k-MACWP. Let $w(p)$ be the weight of a walk p, and $a(p)$ be the average cost of p. We have that $w(p) \leq c$ if and only if $a(p) \leq \frac{c}{k}$.

Hence the minimum average weight s,t-k-walk in G will be the minimum cost s,t-k-walk in G. □

Similarly the following problems can be shown to be NP-hard. We omit the proofs.
Definition 17. k **Maximum Average Cost Simple-Path Problem (k-MaxACSPP):** *Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s,t-k-path p that maximizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of p divided by the number of vertices in p.*

k^+ **Maximum Average Cost Simple-Path Problem (k^+-MaxACSPP):** *Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-k^+-path p that maximizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of p divided by the number of vertices in p.*

k^- **Maximum Average Cost Simple-Path Problem (k^--MaxACSPP):** *Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$, find an s, t-k^--path p that maximizes the average cost of p. The* **average cost** *of a path p is defined as the the total additive cost of p divided by the number of vertices in p.*

k-Minimum Variance Simple-Path Problem (k-MVSPP)*: Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$ such that $(s, t)\neg \in E$, find an s, t-k-path p that minimizes the variance of weights for the set of vertices in p.*

k^+-Minimum Variance Simple-Path Problem (k^+-MVSPP)*: Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$ such that $(s, t)\neg \in E$, find an s, t-k+-path p that minimizes the variance of weights for the set of vertices in p.*

k^--Minimum Variance Simple-Path Problem (k^--MVSPP)*: Given a graph $G = (V, E)$, with positive vertex weights $w(v)$ for each vertex $v \in V$, and vertices $s, t \in V$ such that $(s, t)\neg \in E$, find an s, t-k^--path p that minimizes the variance of weights for the set of vertices in p.*

4.8.4 Dealing with hardness

In Sections 4.7.7 and 4.7.7, we present related work in approximation. In particular, it is not expected that viable solutions exist for minimum/maximum average cost and minimum variance. Using our policy language users can qualitatively eliminate nodes with high values of confidence. This can help find paths with low overall average confidence (the opposite can be done for maximum average). For minimum variance, a user may specify that any link for which the confidence values at its incident routers differ by more than δ is to be avoided. Hence every hop will avoid a large change in confidence, thus providing lower variance. The user can translate their problem into acceptable one-hop precedence properties as an alternative to the more general minimum variable problem. This approach will efficiently yield paths that satisfy the user's modified policy.

4.9 Applications

We present three concrete examples that use our framework. We present the first example in more detail, and suggest two other uses.

4.9.1 High performance and military environments

Consider an MLS (Multilevel Security system) user u_1 with sensitivity level $Confidential$ in compartment $\{Navy\}$, connected at the access point s_1. User u_2 has security clearance $\{Confidential, \{Army\}\}$ and is connected at access point s_{19}. Based on u_1's clearance (u_1 chooses to only reveal this, not its identity), the system presents the user with a logical view of the network as shown in Figure 4.2(a). All routers in this system are cleared for $\{Confidential, \{Army, Navy\}\}$. For simplicity we look at only two inherent attributes: *physical security*, an attribute variable, which can be any value in $\{1, 2, 3, 4\}$. Unshaded nodes represent routers with physical security values 3 or 4, shaded nodes represent values 1 or 2. *Domain* can be $1, 2, 3$ or 4 (we only show D_1 and D_2 in Figure 4.2(a) since we use them as one of the constraints later. These correspond to domains 1 and 2). D_1 can correspond to confidential network owned by the Army for example.

User u_1 desires to communicate with u_2 and determines that u_2's access point is s_{19}. We assume a network component analogous to dynamic DNS which can respond with a user's current access point (u_2 has chosen to register its access point with the service). Now u_1 has been informed by trusted sources that there is an intruder physically located on the premises, and that low physical security routers should be excluded. u_1 specifies the following constraint formula $\mathbf{G_r}$ *physical security* = *high*. This eliminates s_5, s_{15} from the logical view and results in the graph shown in Figure 4.2(b).

The user now wishes to optimize paths based on a unified attribute "x" based on *OS version*, *Domain*, and *physical security*. Under the independent assumption, confidence values of these attributes are unified into one confidence value by multiplying their individual confidence levels. In each case, the confidence value represents the probability with which the user believes the node is not compromised. First, all the confidence values for each attribute at all the nodes are set to 1.

By means of network probes, the u_1 determines the inferred attribute *OS version*, which can be *outdated* (square nodes) or *latest* (round nodes). Routers with outdated operating system versions (*OS version = outdated*) have their confidence levels multiplied by 0.8 since they may be compromised. Lastly, u_1 would like to avoid machines in domain D_1 because of a suspected insider attack in that domain. u_1 multiplies the confidence levels of routers in this domain by 0.4. u_1 has experienced large delays when routing through D_2. Suspecting worm activity, u_1 degrades confidence in those routers by multiplying their confidence levels by 0.7. Figure 4.2(b) shows these confidence levels for each node. Figure 4.2(c) shows the resulting digraph with multiplicative weights. As described in Section 4.7.1 we replace these weight by their negative natural logarithm, and then apply k-shortest path algorithms [HMS03] to obtain the three paths of highest confidence. In this example it is easy to see that the following are paths with the three highest path confidences: $\langle 1, 3, 7, 12, 16, 18, 19\rangle$, $\langle 1, 3, 6, 11, 16, 18, 19\rangle$, and $\langle 1, 3, 6, 10, 9, 13, 17, 19\rangle$. The first two paths have a path confidence of 0.392 (with respect to the logarithmic weights, the total weight is 0.9365), and the third has a path confidence of 0.32 (weight 1.139).

Once these three paths are obtained, the user needs to set up a path through the routers. This is done using a scheme that encrypts the packet multiple times, based on the routers in the path, similar to *onion* routing [RSG98], since public keys of routers are assumed to be well known to u_1. The user encrypts the path in reverse order using the keys of the routers in the reverse path. Each subsequent router decrypts the received route setup packet to obtain the next hop and an encrypted route setup packet for the next router. This technique hides the path from the routers, which only know the previous and next hops in the path. By means of this route setup, u_1 can establish the chosen path to u_2. Packets from u_2 to u_1 are simply forwarded on the reverse path.

4.9.2 Ubiquitous computing

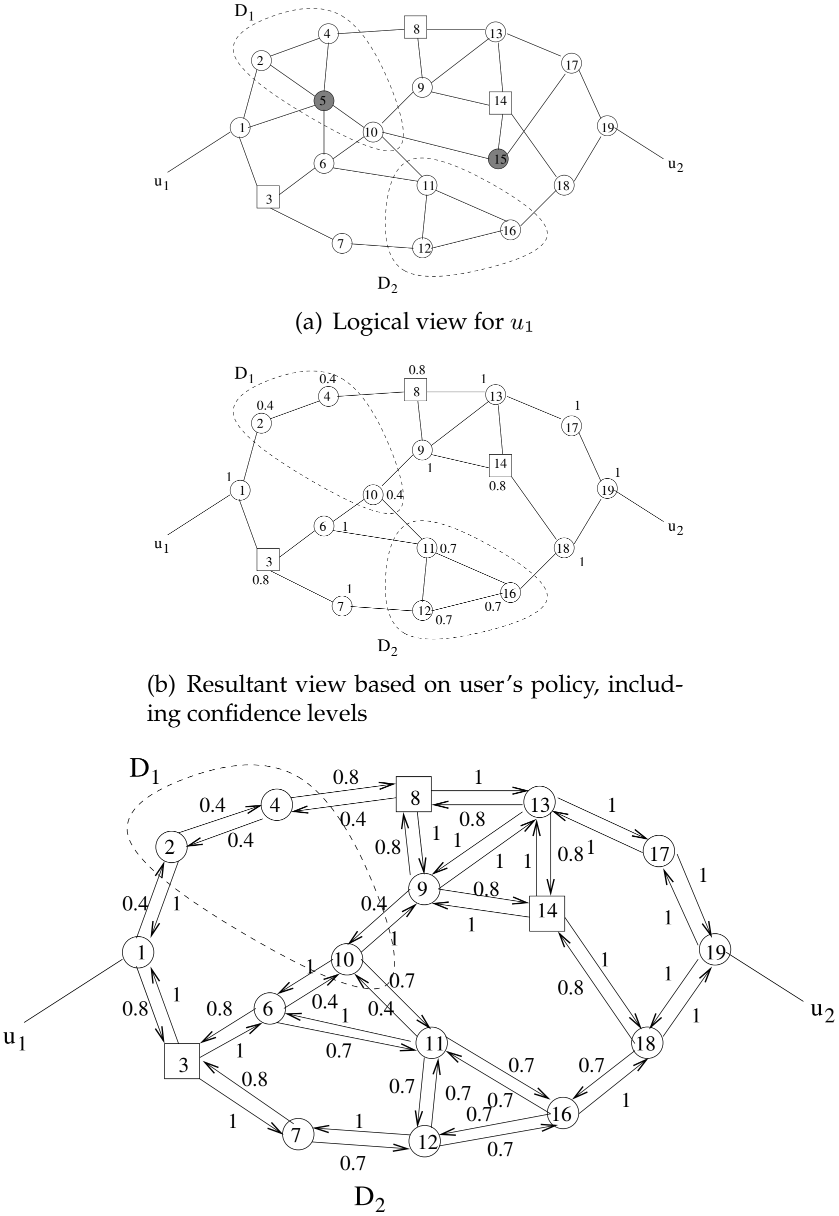
The previous example provided a detailed overview on how our system works in a military environment. In this section we briefly discuss applications to ubiquitous computing. Users in ubiquitous computing environments seamlessly interact with numerous devices and services. In such an environment *discovery* of services, and access to such services is one of the main applications. However, with such an environment it is very easy for the ubiquitous system to track a user's movements or record user patterns. Using our system as a basic infrastructure or service, users can maintain their privacy. Users only need to reveal as much information as necessary to get a logical view of the ubiquitous environment. Again, this is achieved by trust negotiation as described in [YWS03]. In a university setting, a user may want to avoid using routers or services that belong to other research groups, and eliminate these by using global property specifications. While connecting to certain services, a user may choose to maintain location anonymity (for example, using *Mist* [AMCK$^+$02]) by creating a route that is hard to trace back. The user can assign lower confidence levels to the domains that the user does not trust since routers within a domain can presumably collaborate to expose the location of the user. This will give paths of higher confidence that the user trusts for higher location privacy.

4.9.3 Peer-to-peer overlay networks

We consider peer to peer networks where it is feasible for users to obtain topological information of the overlay. We assume the user can form a logical view of the overlay network based on information available to the user. The user desires to perform searches for content available at each peer. Applications include distributed file systems, file sharing, etc. Based on attributes of peers in the overlay, the user can choose to only search for content at routers that satisfy global property specifications. That is, the user can avoid performing searches at untrusted nodes for privacy reasons. Additionally, the user may assign lower confidence levels to nodes on which it expects the search to fail. This can be based on past performance (inferred attribute). If certain nodes seldom have content of interest to the user, the user can assign lower confidence levels. On the other hand, if a biology student is searching for research papers related to cell division, confidence levels for peers in the computer science department can be degraded since there is a very low chance of finding useful content. Hence the user can set up a search path with the highest confidence, so that the probability of the directed search succeeding is high.

4.10 Summary

In this chapter, we argued that users must be given sufficient control over their communication in an organizational network, such as a ubiquitous computing environment. We presented a model that allows users to specify discretionary privacy properties of routes based on the attributes of routers and links in the network based on perceived threat. These attributes can represent boolean as well as real-valued properties. Our policy language allows users to specify global and one-hop precedence properties based on attributes of links and routers. We discussed various representations of trust and showed how trustworthy routes can be computed efficiently for various semantic models of trust. We also identified various models of trust that are computationally hard to satisfy.



(a) Logical view for u_1

(b) Resultant view based on user's policy, including confidence levels

(c) Resultant digraph for use with k-shortest path algorithms

Figure 4.2: Military network example

5 Unlinkability through Access Control

In Chapter 4 we described a model for secure routing. Using this model a user can access services while maintaining location privacy and anonymity. However, when a user accesses multiple services, these various accesses can possibly be linked to the same user. We propose a policy-based framework using RBAC (Role Based Access Control) to address the unlinkability problem in the context of correlating audit records generated from access to distributed services. We explore this problem in an environment such as ubiquitous computing systems, where the enforcement of access control policies is decentralized and ensuring policy consistency as the protection state of the system evolves becomes important. We introduce the notion of an audit flow associated with a user's access transactions, which represents the flow of information through audit logs within an administrative domain. Users of our system can present a set of audit flows to a decision engine that uses global access rules to detect potential linkability conflicts. Users can use this information to specify discretionary unlinkability requirements, depending on whether these accesses can expose sensitive attributes. We present an algorithm that can generate policy constraints based on these discretionary requirements. We also show how these policy constraints can be attached to individual audit log records to enforce unlinkability in a distributed manner. We prove that our proposed algorithm generates constraints that are secure and precise under strong tranquility assumptions with respect to the system's protection state. When we relax these assumptions, we show how versioning can cope with evolving protection state, trading off precision to maintain the security of deployed policies.

This chapter is based on research published in [KNC05].

5.1 Introduction

We examine the problem of preventing administrators from accessing (or "linking") multiple audit records corresponding to transactions initiated by the same user. Our problem is motivated by user identity and location privacy concerns in our university environment, where both physical access[1] to our facilities as well as virtual access to computing resources across our department and the wider campus are controlled by software. While the problem of loss of privacy when a user's actions are observed *in person* by a third party persists, the integration of physical and virtual access control mechanisms exposes a new concern. Users are now worried about other users being able to track their movement as well as their service-access behavior *remotely* by correlating system audit logs. These system audit logs are stored in various databases with independent access control mechanisms, making the enforcement of unlinkability a difficult task. While centralized mechanisms can solve such problems in theory, such approaches are not practical in our setting. They present a bottleneck for distributed access to resources and also provide a single point of failure for access control.

Traditionally, *unlinkability* is defined as the infeasibility of an adversary to correlate two transactions initiated by the same user who does not reveal his/her identity, even when the user presents the same set of attributes to gain access. To address this problem, researchers have proposed a number of cryptographic mechanisms to construct anonymous credentials [CE86; Bra00; CL01; LRSW99] that make it computationally infeasible for a server to link the use of these credentials. However, even if a user presents an anonymous credential to access a service, the set of users allowed to possess those credentials in the first place may be small enough compromise anonymity. Furthermore, while many of these schemes rely on providing user anonymity, there are systems in which users cannot be anonymous. For example, an organization may be required to keep detailed audit records about who accessed payroll information by law. However, access to such information should only be provided to authorized users. In such systems, it becomes important to provide unlinkability through access control, allowing for linkability in only certain cases, e.g., legal subpoenas. We also note that cryptographic mechanisms are also vulnerable to collusion attacks between verifiers and issuers that correlate timing information for access logs [PM04], and adequate access control mechanisms can prevent such attacks.

In this context, we introduce **policy-based unlinkability** as the problem of restricting access to multiple audit records belonging to the same user, corresponding to multiple access transactions, which can be correlated to expose sensitive information. For example, two or more log records that can link a user Alice's identity with her location or other privacy sensitive attributes must not be accessible by administrative users unless they are explicitly authorized by a mandatory system

[1] Users have to swipe their i-card to gain access to the building, labs, offices etc. Each access attempt, along with the user-id and time stamp as well as the access decision, is recorded in a database.

policy or allowed by Alice. Our goal therefore is to provide a framework that can analyze conflicts, and change the authorizations (except when they are explicitly required by system policy) to access audit logs, and prevent such exposures. For example, the system may inform Alice that users in role Network Administrator can access information about her network transactions. There may be some Network Administrators who are also Students and Research Programmers. Based on Alice's privacy requirements, she can then request that Network Administrators who are Students be prevented from linking her audit records. In effect, the system allows Alice to negotiate a set of constraints to prevent certain administrative users from linking her transactions.

In this aspect, our work is related to the Separation of Duty (SoD) problem [SZ97] in the context of RBAC, where different tasks and their associated authorizations are distributed among multiple users to prevent fraud and errors. Our problem is similar in the sense that we are interested in engineering an appropriate set of roles with specific audit access authorizations subject to constraints. However, instead of preventing a user from performing two or more related actions on *single* object in the context of a work flow, our separation of duty problem is about preventing a single user from accessing *two or more* related data objects, across different audit flows. This distinction impacts the way in which unlinkability constraints can be enforced. In Chinese Wall policies, objects are grouped into *Conflict of Interest* (COI) classes and individual users are not allowed to access information from two or more objects in a COI class. Chinese Wall policies and SoD for workflows can be enforced using history-based approaches [San98; Min04]. Since each object can only record its local history and cannot exchange this information with other related objects without explicit coordination, a decentralized approach that does not rely on the access history of individuals or groups is more scalable. The semantics of our approach is more restrictive than the Chinese Wall approach, allowing us to enforce unlinkability in a decentralized way. As with Chinese Wall policies, in our approach we can prohibit administrators from accessing two or more audit-flows in a session; however, we do not guarantee that administrators identified as threats can access individual flows of their choosing. In contrast, Chinese Wall policies allow administrators to access any object in a COI class, but prevent subsequent accesses to other objects in that COI class. This assumption impacts our solution and we prove how our unlinkability policies can be enforced in a decentralized setting without requiring a history of accesses.

The task of building a system that can accommodate for individual users' unlinkability constraints for any ordering of access transactions, with guarantees that extend over long periods of time, is formidable. It is not feasible for the security engineers to simulate all the possible access transaction scenarios for different users, identify unlinkability violations and enforce authorizations accordingly, especially when new services, users and modes of interaction are added periodically. To tackle this problem, we propose a policy engineering framework where a system entity called the Policy Negotiation Server (PNS) works with the user to refine the authorizations to access audit traces based on discretionary unlinkability concerns. The PNS collects information about different audit logs and their associated authorized users and stores it in a policy database. A user Alice can present to the PNS a set of access transactions constituting a session, ask the PNS to analyze these sessions for conflicts, and subsequently update this set by adding new transactions. We explore this problem in the context of RBAC [SCFY96; FK92a] (Role-Based Access Control) system, where users are assigned to roles, which are associated with a set of permissions. The PNS identifies conflicts and generates authorization constraints in terms of access restrictions on user roles, which can be attached to Alice's audit flow records in our system. Using these constraints, accesses to these objects can be allowed or denied based on local access control decisions. With respect to Chinese Wall [BN89] policies that are created with specific threats in mind (e.g., an employee working for two organizations), our analysis *identifies* linkability threats based on the specified access transactions, and allows users to negotiate policies based on these threats.

Assuming that we trust the underlying enforcement mechanisms, given a set of user transactions, we show how we can generate policy constraints that are both secure and precise with respect to enforcing unlinkability properties. We first prove these results under the strong tranquility assumption where the user-role assignments and permission-role assignments do not change over a session. Subsequently, we show how we can relax these assumptions and present an algorithm that uses versioning to handle changes in the authorizations under a weak tranquility assumption, sacrificing precision for the ability to change protection state. Using versioning we can always identify the set of users for which the policies are secure and precise. In both cases we show how users can add new flows to their sessions, refining their unlinkability requirements iteratively.

The rest of the chapter is organized as follows: In Section 5.2 we present an architectural overview of system components and describe the basic policy negotiation protocol. Section 5.3 describes our approach, highlighting the assumptions, formulating the problem, and presenting our algorithm to generate policy constraints in detail. This section also presents a proof of how our algorithm is secure and precise under the strong tranquility assumption. Section 5.4 relaxes this assumption, and presents results for systems with evolving protection state. We summarize this chapter in Section 5.5.

5.2 Architecture

In this section, we provide a brief overview of our system architecture. We introduce some terminology and outline the steps involved in policy negotiation shown in Figure 5.1.

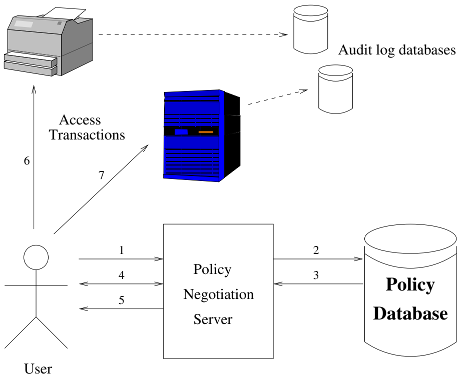


Figure 5.1: System Architecture

We assume a distributed system for sharing resources that allows us to specify and enforce system-wide access control policies. Users in the system access services by presenting credentials resulting in an *access transaction*. We focus on a single administrative domain, where users may engage in attribute-based authentication, possibly with cryptographically unlinkable certificates, although identity-based authentication may be used as well. Users negotiate unlinkability policies with a policy negotiation server (PNS) that generates policy constraints, which when enforced correctly prevents any exposure of sensitive information through audit log analysis by other administrative users.

Information related to an access transaction (e.g., audit logs for location tracking) is stored in one or more databases. We define an *audit flow* for a given transaction as the set of databases and their associated authorizations that define the possible flow of audit information to other users in the organization. A collection of transactions and their associated audit flows that a user desires to keep unlinkable is called a *session*. Sessions are associated with individual users and may be open-ended, i.e., they last for the lifetime of the system and users are allowed to update the list of transactions in their session. As mentioned earlier, users and access permissions are organized into roles using RBAC.

We now describe a high level overview of our system with the aid of Figure 5.1.

(1) In the first step, a concerned user Alice sends her session information to the policy negotiation server (PNS). This is a set of unique identifiers corresponding to access transactions. In steps (2)-(3), the PNS looks up relevant information for each service including access policies and flow policies (replication of data between servers), builds the audit flows $I_1, \cdots, I_n$, and analyzes them for unlinkability conflicts. The PNS presents Alice with a set of roles whose users can access her audit log information from two or more audit flows.

In Step (4) Alice identifies her discretionary unlinkability requirements in terms of roles whose users she wants to prevent from linking her audit information. This approach places the burden of specifying useful privacy policies on Alice. In a real system, Alice can be guided to make an informed choice based on standard organizational policy. The granularity of access to records in a database can be finer, and Alice can specify requirements for specific fields in records. For simplicity of notation, we only model access to a database record as an all-or-nothing permission, and this can be extended to finer-grained permissions as necessary.

As mentioned earlier, the PNS may not be able to enforce some of Alice's choices if there are mandatory access requirements. After Alice and the PNS agree on the policies, in (5) the PNS sends the Alice a certificate with policy constraints for her audit records. This certificate is a digitally signed and can be tagged to Alice's audit data and sent to the databases as the information is generated. The PNS also stores Alice's discretionary policies and session information in the policy database. In Steps (6)-(7), for each access transaction, Alice presents these certificates, which are attached to audit records that make up the audit flows. These policy constraints enforce her unlinkability requirements for the session. We assume that all interactions are cryptographically secured for authenticity, confidentiality, and integrity.

Alice may update her session at any time and introduce new transactions, which may introduce new conflicts. The PNS generates new constraints in this situation and sends the updated credentials to Alice. We assume that Alice has no motivation to delete any transactions from her session. Therefore, we can guarantee that all old policies will still be honored and each successive iteration of our system will be a refinement of the original access restrictions. However, updating policy constraints every time there is a change in protection state can be prohibitively expensive. In Section 5.4, we show how we can maintain security by trading precision for the ability to change protection state, without impacting the privacy guarantees of previously issued policy constraints.

We deliberately choose to limit the functionality of our PNS as a decision engine that generates and distributes access constraints. Users (or agents working on behalf of the user) have to attach these constraints to their audit records. An alternative design would be to centralize policy decision making at the PNS and require that all databases contact the PNS every time an audit record is accessed, to evaluate whether it should be allowed or denied based on the current set of sessions and discretionary policies in the policy database. While this alternative design is more precise with respect to enforcement of the privacy policies, we believe it induces a performance overhead and may create a single point of failure.

Throughout this chapter we will refer to three types of policies. *Flow policies* are explicit representations of data flows between databases. For example a policy such as (d_1, d_2) allows database d_1 to supply copies or transformations of data to d_2. The system can use these flow policies to construct graphical representations of audit flows throughout the system. *Access policies* are Permission-Role assignments (d, r), where role r may access database d. Lastly *policy constraints* are described in Section 5.3.5, and are attached to audit records. Access to an audit record is granted to users based on the access policy for that database, and the policy constraints of that audit flow, which can override the former.

An important point to note in our framework is that we address the unlinkability problem at the level of RBAC access permissions, and assume that the system is aware of the semantics of how the information stored in the audit logs can be linked to expose sensitive user information. However, we do not require that users are aware of the semantic relationships across audit logs. If a user is not aware of the relationships, our system can still enforce a default policy that is conservative and prevent anybody who has access to two or more flows from accessing audit records in both. Techniques such as information hiding, anonymizers and statistical mixing etc., address the unlinkability problem at the semantic level by modifying the contents of the databases. We believe that using access control to enforce unlinkability complements these other approaches and our solution explores the effectiveness of using access control permissions to perform audit flow analysis comprehensively, and prevent linkability in this context.

5.3 Approach

In this section we present our algorithm for constructing audit flow graphs and for generating policy constraints from these graphs based on the system's mandatory requirements and the user's discretionary unlinkability policies. Users in the system typically interact with various services over the lifetime of the system. The set of access transactions that a user wishes to keep unlinkable is referred to as a *session*.

Our first goal is to provide users with a set of mechanisms to negotiate and enforce unlinkability for all their transactions as specified in their session. We explain our system with regard to a particular user Alice who wishes to keep her transaction information unlinkable from other users in the system. For example, Alice may decide that her location information should not be linked with the use of physical services that she may access anonymously. Correlating logs with the location database would expose her identity. Hence, Alice may want to restrict the ability of other users in the system to access both audit flows, viz., location information and Alice's service audit trail. We also show how we can extend this protection when Alice iteratively adds transactions to her session, which we call "open-ended sessions."

One way to specify policy constraints to preserve unlinkability for Alice's session is to find all the users who can access information from two or more flows and restrict their access to sensitive audit records. We propose that data objects in an audit flow are tagged with these access policy constraints in terms of lists of users that are not authorized to view the data. We rely on the trustworthiness of the underlying access mechanisms to enforce these authorizations correctly. Since we assume that the system is working with the users to protect their privacy, access is granted only if these constraints are not violated.

One of the issues with this approach is that these access lists associated with audit flow records could become very large. Furthermore, changes in user permissions during the lifetime of the session will affect the validity of the policy constraints. To improve the compactness of policy representation and handle dynamically changing authorizations, we feel that an RBAC system can address these concerns effectively. In an RBAC system, users are removed and added to roles, and the permissions for roles will not be affected by these operations. Furthermore, our policy constraints can provide instant feedback to administrators as to why their access was denied.

We now propose an approach to enforce unlinkability for Alice's transactions, defined by her session, by finding all possible roles that are explicitly granted read access to each audit flow. We then examine all the users in this set of roles and construct a set of *overlapping roles*, i.e., the set of roles that these users can activate in the system. Roles with a common user are overlapping roles. The main idea here is that if two audit flows have common overlapping roles, then the flows are potentially linkable. A common overlapping role indicates that there may be users with that role who can access both audit flows. We also assume that for the purposes of accessing audit logs, the system has access to all the roles that a user *can* activate, not only those that the user has activated currently.

For example, say role *Administrator* can read audit flow I_1. Now suppose that there are two users u_1, u_2 in the system with the *Administrator* role that also possess the *Student* role, which is an overlapping role. Further, suppose that u_1 in the *Teaching Assistant* role can read audit flow I_2. Since *Student* appears in both audit flow graphs, it is a common overlapping role. Since there is a particular user u_1 in role *Student* who can access both flows, we call *Student* a *conflicting role*. An access policy constraint can be generated if Alice would like to prevent *Students* from linking her flows. Alice could specify "Do not allow any user whose role set contains all the roles $\langle Student, Administrator, TeachingAssistant\rangle$ to access audit flow records tagged I_1 and I_2." While this will prevent *Students* from accessing Alice's information, it will still allow other (non-*Student*) *Administrators* and *Teaching Assistants* to access I_1 and I_2 respectively. Furthermore, students who are not linkability threats (i.e., those who can access only one flow), will still be allowed to access Alice's audit records. A reference monitor enforcing access to the audit record database will check if a particular user has all three roles at the time of access and deny access appropriately. This is more concise than listing all the possible users that can link audit flows. We now formalize these concepts, and show how we can provide users with unlinkability with respect to audit flows. The key idea here is that Alice can specifically deny users of certain roles from linking her information.

5.3.1 Notation

Our system includes roles, databases, and users. In this chapter we refer to these entities both in the context of general access control, and as vertices in graphs. For simplicity, we use the same notation for both contexts, instead of having separate "role vertices" for the corresponding roles, and so on.

5.3.2 Audit Flow Graph

Let the set of roles in the system be Γ, and the set of databases be Δ. Let $\mathcal{U}$ be the set of users in our system. Let URA and PRA be the user-role assignment and the permission-role assignment, defined according to standard RBAC terminology. $URA(u)$ returns all the roles that a user u can activate. Similarly, $PRA(r)$, returns all the permissions or accesses allowed to a role r.

An audit flow graph for an access transaction is a directed graph $I = (V, E)$ with the set of vertices $V \subseteq \Delta \cup \Gamma$, representing databases, roles, and overlapping roles.

Overlapping roles are discussed shortly. A directed edge $(u, v) \in E$ indicates the flow of audit information from u to v. We identify the first database in the audit flow I_i of a given user as the *root* vertex δ_i for that flow.

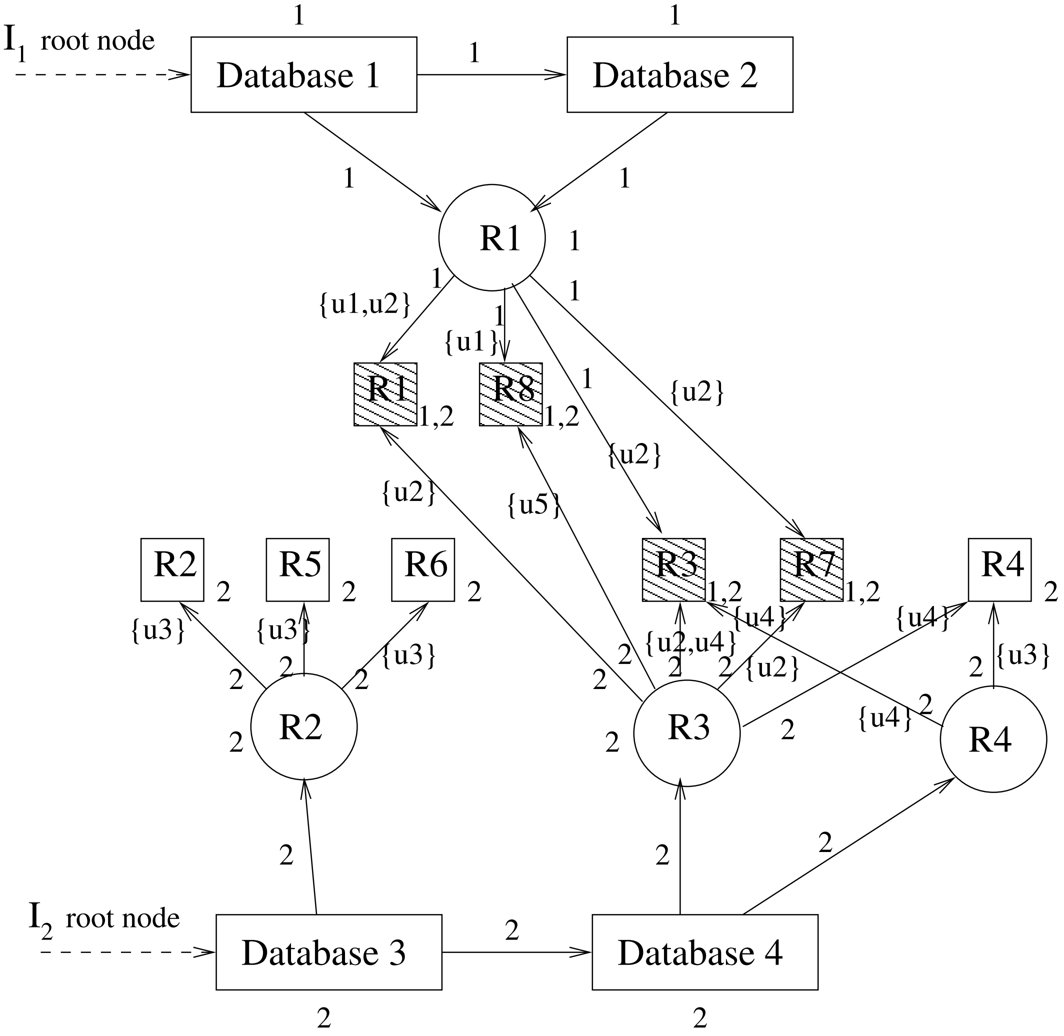


Figure 5.2: Session Graph

We now describe how to create an audit flow graph, given a root vertex that represents Alice's transaction, and show how to construct the combined session graph for multiple audit flows. Figure 5.2 represents an example session graph for two audit flows, which we will refer to for clarity. The audit flow graph I_i for transaction i for user u is constructed as follows:

1. Adding databases: The root vertex δ_i represents the start of the audit flow I_i. Starting from this vertex, we iteratively add vertices and edges corresponding to all databases that receive audit log information about the access transaction δ_i initiated by the user. This operation is repeated until all databases for the audit flow have been added to the audit flow graph. For databases d_1, d_2 we have $(d_1, d_2) \in E$ if and only if the audit flow information for that transaction flows from d_1 to d_2.

In Figure 5.2, databases are represented as rectangles. The root vertex for I_1 is *Database 1*. As information related to audit flow I_1 flows from *Database 1* to *Database 2*, we have a directed edge from *Database 1* to *Database 2*. Similarly, we have audit flow I_2 flowing from *Database 3* to *Database 4*.

2. Adding roles: For each database $d \in V$, determine the set of roles $R \subseteq \Gamma$ with read permission to database d. These roles are added to the audit flow graph vertices V, along with the edges (d, r) for each $r \in R$. We have $(d, r) \in E$ if and only if role r has permission to read database d.

In Figure 5.2, roles are represented as circles. The individual access policies of *Database 1* and *Database 2* allow read access to users with role $R1$. Hence we have directed edges from *Database 1* and *Database 2* to $R1$, and so on.

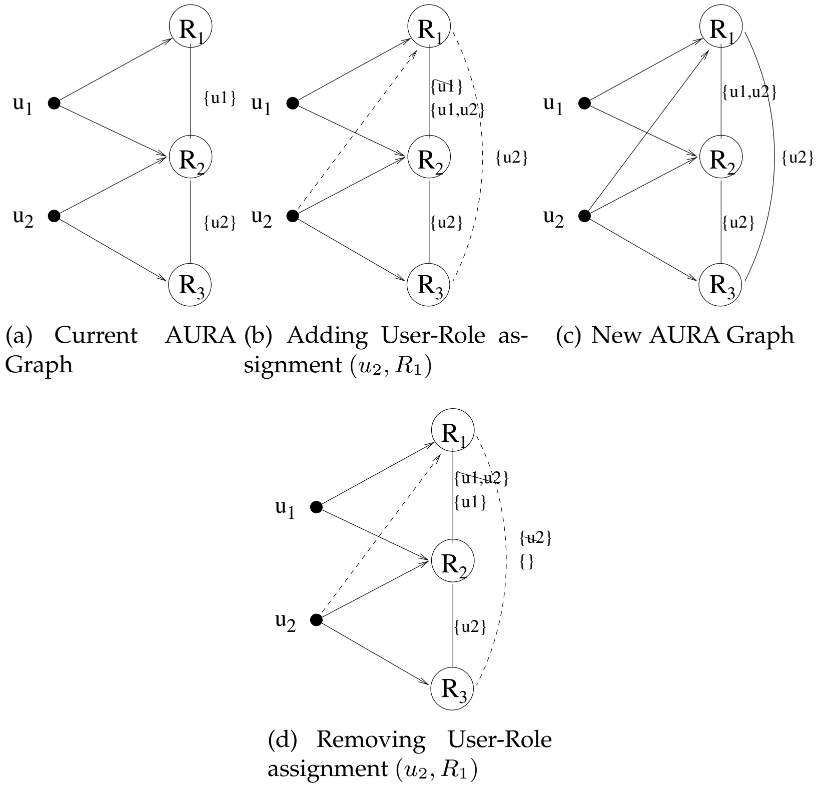
3. Adding overlapping roles: For each role $r \in V$, we generate the corresponding overlapping roles, and include directed edges to them. Let $O \subseteq \Gamma$ be the set of overlapping roles such that for every $o \in O$, some user u can activate role o in addition to r. We call r the *parent* of overlapping role o. We have $(r, o) \in E$ if and only if o is an overlapping role of r.

Consider the following URA for a system with five users u_1, u_2, u_3, u_4, u_5. $URA(u_1) = \{R_1, R_8\}$, $URA(u_2) = \{R_1, R_3, R_7\}$, $URA(u_3) = \{R_2, R_5, R_6\}$, $URA(u_4) = \{R_3, R_4\}$ and $URA(u_5) = \{R_3, R_8\}$ Figure 5.2 shows the overlapping roles overlapping roles (represented as squares). Role R_1 has overlapping roles $\{R_1, R_3, R_7, R_8\}$, R_2 has overlapping roles $\{R_2, R_5, R_6\}$, and so on. The user-sets on edges of overlapping roles show the users common to both roles.

We now examine the complexity of creating an audit flow graph for a given transaction. In Step 1, at most $|\Delta|$ new vertices can be added to the graph. For each vertex, at most $|\Delta| - 1$ new edges can be added. Therefore we are bounded by $O(|\Delta|^2)$ operations. In Step 2, for each database, at most $|\Gamma|$ role edges can be added to the graph. Therefore Step 2 is bounded by $O(|\Delta||\Gamma|)$ operations.

Constructing the AURA Graph

Step 3 involves generating overlapping roles. We show how we can amortize the cost of this step by augmenting a standard URA mapping to include overlapping role assignments. We call this the *AURA* graph (Augmented User Role Assignment graph). A directed edge (u, r) mean that user u is assigned to role r. An undirected edge (r_1, r_2) means that r_1 and r_2 are overlapping roles. Each undirected edge is associated with the set of overlapping users for roles r_1 and r_2, $U(r_1, r_2)$. In Section 5.3.3 we will use these user-sets to identify conflicting roles. A *conflicting role* is a role that contains one or more users who can access two or more audit flows within a session.



(a) Current AURA Graph (b) Adding User-Role assignment (u_2, R_1) (c) New AURA Graph

(d) Removing User-Role assignment (u_2, R_1)

Figure 5.3: AURA Graph example

We show how we can use the AURA graph to maintain overlapping role information, and describe how to update an AURA graph when the protection state of the system changes:

Adding a Role: This operation does not create any extra overhead, since overlapping roles are not affected until a User-Role assignment changes.

Adding a User-Role assignment: If a User-Role assignment (u, r) is added, then for each of u's roles $r' \in URA(u)$, the undirected edges (r, r') are added unless these edges exist already, and u is added to the set $U(r, r')$. There are $|URA(u)|$ operations, which is bounded by $|\Gamma|$. The time complexity for set union for adding u to $U(r, r')$ is constant (using hash tables). For example, consider the AURA graph in Figure 5.3(a). We omit self-loops (r, r) with user-sets $U(r, r)$ equal to the set of all users in r. We add the assignment (u_2, R_1) as shown in Figure 5.3(b). We must now update edges (R_1, R_1), (R_1, R_2) and (R_1, R_3), resulting in three operations on the AURA graph. Since $URA(u) = \{R_1, R_2, R_3\}$, we have $|URA(u)| = 3$ as expected. The resulting AURA graph is shown in Figure 5.3(c).

Removing a User-Role assignment: If a User-Role assignment (u, r) is removed, then for each of u's roles $r' \in URA(u)$, u is removed from $U(r, r')$. If $U(r, r') = \emptyset$, the edge (r, r') is removed. There are $|URA(u)| + 1$ operations, which are bounded by $|\Gamma|$. Again, removing u from $U(r, r')$ can be done in constant-time with the use of hash tables. In our previous example we added the assignment (u_2, R_1), resulting in the AURA graph shown in Figure 5.3(b). To remove this assignment, we must update the edges (R_1, R_1), (R_1, R_2) and (R_1, R_3), as shown in Figure 5.3(d). Here $URA(u) = \{R_2, R_3\}$ since the assignment (u_2, R_1) was removed, and we have $|URA(u)|+1 = 3$ as expected. The resulting AURA graph is the same as the original AURA graph in Figure 5.3(a).

Removing a Role: Each User-Role assignment must be removed first. Let U be the set of users for the role being removed. Hence we have $|URA(u)|$ operations for each user $u \in U$. This is bounded by $|\mathcal{U}||URA(u)|$.

Therefore, this approach requires at most $|URA(u)|$ operations on the AURA graph for each addition/deletion of a User-Role assignment. In the worst case this is $O(|\Gamma|)$ operations for each addition/deletion of a User-Role assignment. Deleting a role in the system is more expensive and is bounded by $O(|\mathcal{U}||\Gamma|)$. Overlapping roles for any particular role can be efficiently extracted from the AURA graph by a simple lookup.

5.3.3 Session Graph

While individual audit flow graphs capture the dissemination of log information to authorized users in an organization, users are interested in exploring how their sensitive attributes can be exposed by log correlation across these databases. Given a set of audit flows $\{I_1, \cdots, I_n\}$, corresponding to a set of transactions that user Alice may execute, we define session graph S by constructing a composite graph which includes each audit flow graph that was constructed as described in Section 5.3.2. The set of vertices and edges in the composite graph is the union of the sets of vertices and edges in the original audit flow graphs. However, we preserve the information about distinct flows in this composite graph by augmenting edges with colors as described next.

In order to represent overlapping nodes and edges between these graphs and identify linkability conflicts, we introduce the mapping $Color : I_i \rightarrow \mathbb{N}$, which identifies a unique natural number with each audit flow. For simplicity, we assume that edges $e_i \in E_i$ from I_i are assigned color i, i.e., $Color(I_i) = i$. An edge $e_s \in S$ may therefore have multiple colors, reflecting which flow it belongs to for each color. We define the colors for a vertex $v_s \in V_S$ as $Colors(v_s) : V_S \rightarrow 2^{\mathbb{N}}$, as the set of colors of its incident edges. Figure 5.2 shows the session graph with colors for each edge and vertex.

Let $C' \subset V_S$ be the set of all overlapping role vertices in the composite session graph S with two or more colors. We call this the set of *potentially conflicting roles*. These roles may contain users that have static read access to two or more flows. To illustrate, R_7 and R_8 are potentially conflicting roles in Figure 5.2, and are indicated with shaded squares. After these potentially conflicting roles are identified, they are further examined for linkability conflicts.

Consider the potentially conflicting role $c' \in C'$. Recall that all the incident edges (r, c') are augmented with the set of common users $U(r, c')$ from the AURA graph, in addition to their colors. For a given potentially conflicting role, if the intersection of the user sets for edges of *different* colors is not empty (that is if there is a user u in two edge sets of different colors) then we identify c' as a *conflicting role*. Also, if any edge has two or more colors, and at least one user in its user-set, then c' is a conflicting role. Let the set of conflicting roles be $C \subseteq C'$.

In Figure 5.2, R_8 is not a conflicting role since there are no users in R_8 that are in parent roles R_1 and R_3, that can access flows of different colors, viz., I_1 and I_2. R_7 is a conflicting role because u_2 appears on the edges (R_1, R_7) and (R_3, R_7), i.e., user u_2 with role R_7, also has roles R_1 and R_3 and can access two flows of different colors I_1 and I_2. The conflicting roles in Figure 5.2 are R_1, R_3 and R_7.

Complexity of detecting conflicting roles: Let E be the set of incident edges on a potentially conflicting role c'. In the worst case, each edge $e \in E$ has a different color from the other edges. For each color i (or flow), compute the union $\mathcal{U}_i$ of the edge sets $U(r, c')$ for all parent roles r of c' and all edges with color i. $\mathcal{U}_i$ is the set of users in c' that can access flow I_i. Now we must check for pairwise intersections between the $\mathcal{U}_i$'s ($O(n^2)$ intersections) to identify real conflicts. Since there are at most $|E|$ union operations bounded by the number of roles $|\Gamma|$, and each such operation is linear in $|U(e)|$ bounded by $|\mathcal{U}|$ (set union using a hash table), the worst case complexity for this step is $O(n^2|\mathcal{U}| + |\Gamma||\mathcal{U}|)$.

We now show how a user of the system can specify discretionary policies representing unlinkability requirements and present an automated technique to generate constraints on the dissemination of audit flow information. We also show how if we can enforce these constraints appropriately, we can prevent linkability.

5.3.4 Specifying discretionary policies

As described in Section 5.3.3, the PNS returns to Alice a set of conflicting roles C in S. Alice picks a subset of these roles C_{Alice} as her discretionary unlinkability requirements.

A linkability conflict occurs for users with role $c \in C_{Alice}$ that can access a database belonging to two or more flows. When Alice creates a new audit record that flows to a database that can be accessed by a user in a conflicting role, the underlying access control system denies the right to access these records to all users in these roles who pose a linkability threat. The PNS subsequently generates policy constraints that Alice can attach to her audit records.

5.3.5 Generating and enforcing policy constraints

We call C_{Alice} Alice's *deny-set*. The members in Alice's deny-set should be prevented from linking Alice's flows. Note that not all users in the deny-set are linkability threats, and hence we need to make sure that only the users who can link Alice's flows must be denied access. We define the Alice's policy constraints for session S, $\mathcal{P}_S$, as the tuple $\langle C_{Alice}, \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$, where $\mathcal{R}_i$ is the set of roles with static read permission to information flow I_i, and are parents of some role in C_{Alice}. This is easily obtained from the session graph S.

Audit flow records in session S are tagged with $\mathcal{P}_S$. When a user u attempts to access an audit record, the database's reference monitor first checks to see if u has static read access for that database. If so, it then checks the attached $\mathcal{P}_S$ to see if any of u's roles are in C_{Alice}. If so, the reference monitor checks to see if u's role-set $URA(u)$ has a non-empty intersection with at least two different sets in $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$. If so, the user has static read access to two or more flows in S, and the user is denied access by the reference monitor. In the worst case, for users with static read access to the database, the reference monitor needs to compute $n + 1$ intersections, where each intersection takes $O(|URA(u)| + |\Gamma|)$ operations, which is $O(|\Gamma|)$. Hence the time complexity for evaluating $\mathcal{P}_S$ is $O(n|\Gamma|)$ if u is in Alice's deny set. If not, the time complexity is $O(|\Gamma|)$, the cost of computing the intersection $URA(u)\hat{C}_{Alice}$. From Figure 5.2, assuming that $C_{Alice} = \{R_7\}$. We have $\mathcal{P}_S = \langle \{R_7\}, \{R_1\}, \{R_3\} \rangle$.

At this point, a valid question is why not generate policy constraints with user IDs. There are three reasons for this. Firstly, if a user u was identified to be a linkability threat, then adding u to the policy constraints will prevent u from accessing two or more flows. However, if u is removed from a particular role and is no longer a linkability threat, u will *still* be denied access. Our scheme adds more precision to the system by allowing users who are no longer linkability threats to access audit records. Secondly, we would like to give administrators feedback as to why their access was denied. Our policy is able to capture the reasons *why* access control decisions are made in addition to *what* access control decisions are made. And lastly, in large systems we expect a role based formalism to be a more compact representation of linkability conflicts.

We now present two definitions, and prove that our system is secure, sound, and precise under certain assumptions.

Definition 18. *If the access permissions for a database record associated with a flow for user* u *includes the right to read, then we say that* u *has* **static read access** *to the audit flow. These static permissions can be overridden by policy constraints.*

Definition 19. **Strong Tranquility** *asserts that the access permissions associated with the users of the system (i.e., the URA and the PRA) do not change by system operation.*

Policy constraints are generated based on the current protection state of the system (i.e., the URA and the PRA). Changes to the protection state can result in policy constraints that are "out of date." We first prove that our constraints are *secure*, *sound*, and *precise* under the strong tranquility assumption. We relax this assumption in Section 5.4 and show how we can trade precision for security when the protection state and the session information are allowed to change. The following theorems are easy to prove because of the strong tranquility assumption, which makes the properties hold by construction of session graph S and policy constraints $\mathcal{P}_S$.

Theorem 11. *user u has static read access to two or more audit flows in a session, then all of the user's roles $URA(u)$ appear as conflicting roles in the session graph.*

(Security) *Assuming strong tranquility, if a user* u *with a role in Alice's deny-set C_{Alice}, has static read access to two or more audit flows in Alice's session $I_1, \cdots, I_n$, the policy constraints will prevent* u *from accessing these flows. Furthermore, Alice was presented with all of u's roles as conflicting roles.*

Proof. Since u has static read access to two or more flows in $I_1, \cdots, I_n$ and since we assume strong tranquility, by construction all of u's roles will appear as conflicting roles in the session graph S. By construction of the constraints, u will be denied access to $I_1, \cdots, I_n$.

□

Theorem 12. **(Soundness)** *Assuming strong tranquility, if a user* u *is denied access to a flow I_i by the policy constraints, then the user has static read access to two or more audit flows in the session S.*

Proof. Since u was denied access by the policy constraints, u's role set includes a conflicting role role $c \in C_{Alice}$, and intersects with two or more role sets in $\mathcal{R}_1, \ldots, \mathcal{R}_n$. Since we assume strong tranquility, this implies that u has access to two or more flows in S.

□

The following theorem is simply the contrapositive of Theorem 12. In the following sections we will only refer to security and precision, since precision follows from soundness.

Theorem 13. **(Precision)** *Assuming strong tranquility, if a user* u *has static read access to exactly one audit flow within a session, then* u *is not denied access by the policy constraints.*

5.3.6 Open-ended sessions

Our algorithm in Section 5.3.5 maintains security and precision for a predefined session. Consider the case when user Alice does not know all her transactions *a priori*. Alice would like to dynamically generate constraints for new audit flows, without invalidating her constraints to older audit flows. We extend our algorithm to allow users to add audit flows to existing sessions and generate new constraints appropriately.

Consider the session graph S, and the new flow I_{n+1}. Construct the session graph S' by combining the audit-flow graph for I' with S as described previously in Section 5.3.3, and generate the new policy constraints for audit-flow I_{n+1} as described in Section 5.3.5. We now show how security and precision holds for session S'. We modify the definition of security to allow access to at most one flow, since this does not violate unlinkability, and implies the security property defined in Theorem 11.

Theorem 14. **(Security)**

Assuming strong tranquility, if a user u *with a role in Alice's deny-set C_{Alice}, has static read access to two or more audit flows in Alice's session $I_1, \cdots, I_{n+1}$, then the policy constraints will prevent* u *from accessing* **two or more** *of these flows.*

Proof. We prove this by induction on the number of audit flows. For the base case we consider policy constraints generated for one audit flow. The set of constraints is empty. Since there is only one flow, there are no linkability conflicts. Now consider session S with audit-flows $I_1, \ldots, I_n$, and assume the security property holds for policy constraints for flows in S. If we generate new policy constraints for I' as described in Section 5.3.6, then any user u that has static read access to two or more flows in S' is denied access to audit-flow I'. Users with static read access to two or more flows in S are allowed access to at most one flow in S (inductive hypothesis). Consider a user u that has static read access to exactly one flow in S, and to I'. Policy constraints for S will still allow u to access a single flow in S, and the new constraints for I' will prevent u from accessing I'. Hence u can access at most one flow in S' and security holds. □

Theorem 15. (Precision)

Assuming strong tranquility, if a user u *has static read access to exactly one audit flow within a session, then* u *is not denied access by the policy constraints.*

Proof. For the base case, again consider one audit flow. Since there are no policy constraints, u will not be denied access by the policy constraints. Assume that for a session S with audit-flows $I_1, \ldots, I_n$, precision holds for the policy constraints. If we generate new policy constraints for I' as described in Section 5.3.6, then any user u who has static read access to exactly one audit-flow in S', will still be allowed access to I'. Consider the case when u tries to access a flow in S. If u has static read access to a flow in S, then precision holds by the inductive hypothesis. If u has static read access to I', then u does not have static read access to any flow in S and is (trivially) denied access to a flow in S.

□

5.3.7 Mandatory audit flows

The PNS may consider access by certain conflicting roles to be mandatory. In our example mentioned earlier, the PNS may mandate that student administrators cannot be denied access (in this case, *Administrator* is the parent role of the overlapping role *Student*). Specifically, the PNS can specify edges (r, o) that are mandatory, where r is a role vertex, and o is an overlapping role of r. Hence, any user with roles r and o are exempted from the policy constraints. If there are exempted users that can access two or more audit flows, the user is informed of this.

Our goal is to make the privacy implications of sensitive information explicit to the user. Users will have complete information of who can access the user's information, and will proceed only if they agree to the PNS's mandatory policy.

In the next section, we relax the strong tranquility assumption and present a discussion of what policies we can enforce when the permissions are allowed to change and investigate the trade-off between security and precision.

5.4 Security under weak tranquility

Our strong tranquility assumption in Section 5.3.5 is restrictive since the users, roles, and permissions, which define the protection state in any organization will change over time. Once the protection state changes, it may not be possible to enforce some of the unlinkability requirements. New conflicts may emerge that may invalidate existing guarantees.

In this section, we extend our results to model the effect of changing the protection state. Our proposed solution uses versioning to localize the impact of these updates. Since our policy enforcement mechanisms are decentralized, i.e., records belonging to a particular flow in a database are tagged with access restrictions, it is important to guarantee the security of these access restrictions under evolving protection state.

We define the notion of weak tranquility which captures the effect of changing permissions on the satisfaction of unlinkability properties.

Definition 20. **Weak Tranquility** *states that the access permissions (i.e., the URA and the PRA) associated with a user* u *of the system do not change in such a way that it violates the security and precision of the enforcement of discretionary unlinkability policies for that user.*

Our goal is to guarantee that changes to the protection state can preserve the weak tranquility property for as many users as possible during the lifetime of the system.

When a policy is agreed upon by the user and the PNS, the policy constraints certificate is stamped with what we call the *system version number* maintained by the PNS. When users are added to the system, they are also stamped with the current system version number. The user's version number will be updated when certain changes are made to the protection state. A user u can access an audit record belonging to flow I only if $Version(u) \leq Version(I)$. We assume that reference monitors have access to the current version number for a user (e.g., policy database or a revocation-based certificate approach). We prove Lemma 6 based on the following update rules for a user's version number.

Lemma 6. *Consider audit flows $I_1, \ldots, I_n$ in a session S. After any change to URA or PRA, if for a user u, $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$, then weak tranquility holds for user u with respect to audit flows $I_1, \ldots, I_n$.*

Proof. We prove this for each possible update to the protection state, and hence the lemma holds by induction on the number of updates to the protection state. For the base case, there are no updates to the protection state, and the lemma trivially holds by strong tranquility, which implies weak tranquility.

New User u Created: No change to system version number. Assign current system version number to user u. u has not been granted any new permissions and weak tranquility holds for u.

New Role r Added: No change to system version number. No permissions have changed in the system, and weak tranquility holds for all users.

User-Role (u, r) Assignment Added: When a User-Role assignment (u, r) is added, it is possible that u now has static read access to two or more flows in session S, but will not be denied access to two or more flows by the policy constraints. To maintain the security property of the policy constraints with respect to u, the system version number is incremented, and u is assigned the new version number. Since the permissions of all other users remain unchanged, security and precision of the constraints hold for all other users, whose version numbers remain unchanged.

User-Role Assignment (u, r) Deleted: No change in version number. We only need to examine the case when u had static read access to two or more flows in S before the user-role assignment was deleted. If u continues to have static read access to two or more flows in S, then u must activate roles other than r, which must appear in the original policy constraints. Hence u will be prevented access by the policy constraints if u has a role in the deny list of the constraints (security property). If u does not have any roles on the deny list (see discussion for *privilege escalation* for the case when $r \in C_{Alice}$), then u is allowed access. If it is the case that u no longer has static read access to two or more audit flows, then r was necessary for access to two or more flows. Hence $r \in URA(u)$ is a necessary condition for being denied access by the policy constraints. Since now $r \notin URA(u)$, the policy constraints will allow u to access flows in S (precision). Since the permissions of all other users remain unchanged, security and precision of the constraints hold for all other users, whose version numbers remain unchanged.

User u Deleted: Version number does not change. Equivalent to iteratively removing all User-Role assignments for u. Delete all the User-Role assignments.

Role r Deleted: Equivalent to iteratively removing all User-Role assignments for r followed by removing all $PRA(r)$. Note that after this operation, the system version number remains unchanged.

Permission-Role (d, r) Assignment Added: This means that a role r has been granted static read access to some database d. Since this role may not have been included in the session graph, it is possible that some users in r can now access two or more audit flows, and will not be denied access by the policy constraints, violating the security of the policy constraints, and weak tranquility does not hold. If there are any users assigned to role r, the system version number is incremented, and all users in r are assigned the new version number. Hence, if $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$, then u is not a member of r, and weak tranquility holds for u.

Permission-Role (d, r) **Assignment Deleted:** This means that the static read access to database d has been removed for a role r. It is possible that users in r are no longer a threat to linkability, but will still be denied access by policy constraints, violating the precision of the policy constraints. Hence weak tranquility does not hold for users in r. If there are any users assigned role r, the system version number is incremented, and all users in r are assigned the new version number. Hence, if $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$, then u is not a member of r, and weak tranquility holds for u. Note that the security of policy constraints is not affected by adding the assignment (d, r). However for every policy we would like to maintain the set of users for which weak tranquility holds, which is why we update the version numbers for affected users.

Privilege Escalation: Consider the situation when a user has access to only one flow in a session. After accessing this information, the user is removed from a particular role, and then added to a new role, giving the user access to another flow in the session, violating the unlinkability requirement. However, the version number of the user is incremented when a new user-role assignment is added, which will prevent this kind of privilege escalation. Similarly, incrementing the version number on the addition of a new permission-role assignment prevents privilege escalation due to changing permission-role changes. More generally, privilege escalation is prevented by the fact that a user's version number is incremented whenever the user's static permission set increases. It is important to note that if a role r is removed from a user's role-set, it is possible that r is on the deny list of some policy constraint, and that the user will now be able to link flows in that session, which was disallowed before this removal. With cooperation from the security officer, a user can remove, and subsequently add, r to his/her role-set resulting in one form of privilege escalation. We assume that the security officer is trusted, and that privilege escalation from the removal of a conflicting role is semantically correct and secure. An alternative approach would be to define this type of privilege escalation as not secure, and increment the version number when a user-role assignment is removed.

$\square$

Under versioning, the following theorems follow from Lemma 6.

Theorem 16. **(Secure)** *If a user u with a role in Alice's deny-set C_{Alice}, has static read* *access to two or more audit flows in Alice's session $I_1, \cdots, I_{n+1}$, then the policy constraints will prevent* u *from accessing* **two or more** *of these flows.*

Proof. If $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$ then the weak tranquility assumption holds by Lemma 6, which implies security with respect to user u. If $Version(u) > Version(I_i)$ then the user is trivially denied access, even if their access did not cause a linkability conflict. □

Theorem 17. (Precise up to Versioning) *If a user u has static read access to exactly one audit flow within a session $S = \{I_1, \ldots, I_n\}$, then u is not denied access by the policy constraints if $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$.*

Proof. If $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$ then the weak tranquility assumption holds by Lemma 6, and hence the constraints are precise up to versioning. For users with higher version numbers, precision does not hold, since they will be denied access even if they cannot link flows within a session.

□

After the policy constraints have been generated, previously deployed policy constraints gradually lose precision by being overly restrictive to users affected by evolving system permissions. However, this is restricted only to users who gain new permissions, and users of roles for which database permissions change. We argue that the latter case is rare and can be performed at predefined system epochs. To cope with degrading precision, the PNS can choose to honor the policy constraints for a certain time-period called *unlinkability window*. This window can either be a static parameter in the system, or can be negotiated with the user. As mentioned earlier, changes in flow policies are considered to be non-trivial changes. These changes can take place in epochs that honor the unlinkability window. When this is not possible, all data along the new flow is tagged as sensitive and is only allowed access by designated administrators. Users can be informed in general that changes in flow policy are possible, and that certain designated administrators will have access to audit flows in the session.

5.5 Summary

In this chapter, we explored the problem of user unlinkability in the context of correlating audit data. Our work examined how administrative users with authorizations to view audit records across different servers in an organization can link different access transactions to sensitive attributes of other users such as identity and location. We showed how this problem persists even if users employ anonymous credentials to gain access to a service. To the best of our knowledge, our work in this paper is the first to discuss a policy-based approach for enforcing unlinkability.

We formalized the unlinkability problem by defining the the notion of an audit flow associated with a user's access transaction. Audit flows for different access transactions can be composed to generate a session graph that encodes the linkability conflicts (potential threats) compactly and captures the scope of the problem adequately. Using this session graph, we showed how we can transform the unlinkability problem into a policy engineering problem, and presented an algorithm to generate authorization constraints that can enforce unlinkability based on a user's perceived threat.

With appropriate tranquility assumptions on the underlying authorizations, we proved that our constraints can guarantee unlinkability. We formalized the notion of security and precision with respect to enforcing unlinkability constraints. To maintain the security of deployed policy constraints under evolving protection state, we proposed a solution based on versioning that maintains security by trading precision for evolving protection state. Using our approach, the set of users for which policy constraints are secure and precise can always be identified.

6 *Know* Why Your Access Was Denied

In Chapter 4 we described a model for trustworthy routing. Section 4.5.2 described how a user is presented with a logical view of the network based on the credentials presented to the system. In general, a user will engage in trust negotiation with the system until he/she is satisfied with these authorizations. If a user desires access to more routers, the system must aid the user by providing authorized feedback on which additional credentials are needed.

In this chapter, we examine the general problem of providing useful feedback about access control decisions to users while controlling the disclosure of the system's security policies. Our model combines the use of qualitative policies and quantitative cost functions to provide useful and authorized feedback. Relevant feedback enhances system usability, especially in systems where permissions change in unpredictable ways depending on contextual information. However, providing feedback indiscriminately can violate the confidentiality of system policy. To achieve a balance between system usability and the protection of security policies, we present *Know*, a framework that uses quantitative cost functions to provide feedback to users about access control decisions. *Know* honors the qualitative policy protection requirements, which are represented as a meta-policy, and generates permissible and relevant feedback to users on how to obtain access to a resource. To the best of our knowledge, our work is the first to address the need for useful access control feedback while honoring the privacy and confidentiality requirements of a system's security policy.

Research on *Know* was done in collaboration with Geetanjali Sampemane [KSC04].

6.1 Introduction

When a user is denied access to a resource, what level of feedback should the system provide? At one extreme, the system can conceal all policy information and respond with a simple "Access is denied." Many security systems use this policy on the grounds that an unauthorized user must not be given any further information. However, such a system is less user-friendly to legitimate users, because they are not given enough feedback to reason about, and correct errors. This problem is exacerbated when access also depends on contextual information, and permissions change based on factors like room activities and time-of-day—without feedback, a user has no way of discerning why some attempts succeed and others fail. At the other extreme, the system can be completely open about its policies and make them public knowledge. Users can then reason about their access to resources, making such a system more usable. While feedback to legitimate users is often a desirable feature, unrestricted feedback can be harmful, for example, by assisting intruders probing system security in finding out where to direct their attacks. Another problem with open policies is information leakage—policies contain enough information to allow legitimate users to deduce what other system users can access, thus violating the privacy of those other users.

As systems get more complex, access policies get more complicated too, and it is unreasonable to expect users to memorize all the conditions that affect access. It thus becomes important to provide useful feedback to legitimate users, while also addressing privacy and security concerns of the system's policies (i.e., providing suitable *policy protection*). Any such system makes a trade-off between usability and policy protection. Providing useful feedback also improves system *availability* to legitimate users. While this issue has not been adequately researched, common operating systems do provide primitive forms of policy protection. For example, UNIX allows users to look at a file's permissions (or access policy) if they have *execute* access to that directory. However, for two files in the same directory, there is no way to hide the policy for one file and not the other. While this coarse-grained policy protection has been adequate in conventional systems, we believe that new ubiquitous computing [Wei91] environments accentuate the problem of policy protection and feedback.

Users in ubiquitous computing environments typically interact with a plethora of computing, communication or I/O devices in their vicinity in many ways—voice, gestures, and traditional keyboard-and-mouse input being some of them. Different sets of users are allowed access to different subsets of resources, and these permissions may change depending on contextual information such as the time of day, the current activity, or the set of people involved. In such an environment, it may not be clear to a user why he or she was denied access to certain resources. Thus, informative feedback about why access was denied becomes very important if the system is to avoid annoying users with apparently-arbitrary restrictions. However, as mentioned earlier, unrestricted feedback about who is allowed to do what in the system could itself compromise system security and privacy; therefore, policies need to be protected against inappropriate disclosure. As a first step in this direction, we present a feedback model called *Know*, which uses *meta-policies* for policy protection and *cost functions* to compute useful feedback.

Know examines the meta-policy that protects a policy and determines the level of feedback that can be provided to a particular user. For example, a student trying to access an audio device in a conference room may be told to return after the ongoing meeting. However, a meeting participant may be informed that the meeting chair has access to the audio device. The basic challenge in providing informative feedback is to identify the conditions required for the requested operation to be allowed, i.e., find a way to "satisfy" the access control rule guarding that operation. This may involve the user activating a different role (or presenting a different credential) or waiting for the context to change (e.g., the current activity configuration of the space). Searching all the rules that guard a particular action will determine all the situations in which this operation is permitted. *Know* can use this information to suggest viable alternatives. A cost function is used to represent the relative difficulty of changing an attribute to satisfy an access condition—it may be easier for a student to wait for the end of a meeting to be allowed access to a printer, rather than to become a room administrator to print the document immediately.

Satisfiability is an NP-complete problem in general. However, we explore the state-space using ordered binary decision diagrams (OBDDs) [Bry86], which are efficient and compact representations of boolean expressions in general, and search for conditions that satisfy the access rules. OBDDs are graph structures, which makes it easy to apply cost functions and shortest path algorithms for providing useful feedback. While certain degenerate cases can result in exponentially large OBDDs, there are several heuristics that usually reduce the size of OBDDs. To improve performance, the OBDDs are computed in advance. *Know* computes feedback only if it can do so within reasonable bounds of time and space.

This work is an initial attempt to address the problem of providing useful feedback about security decisions in ubiquitous computing systems while honoring protected policies. Our main argument is that system feedback is more useful if it is tailored to the intended recipient and based on his or her current permissions rather than a generic "Access is denied" message, and that it is possible to provide such feedback while still protecting policy information appropriately.

The rest of the chapter is organized as follows—Section 6.2 discusses some necessary background and Section 6.3 describes the system architecture including the use of OBDDs and cost functions to efficiently generate feedback. In Section 6.4, we describe our implementation of *Know* and show how it presents useful feedback for a realistic policy. We then discuss some related work and issues raised by our approach in Section 6.5 before concluding in Section 6.6.

6.2 Background

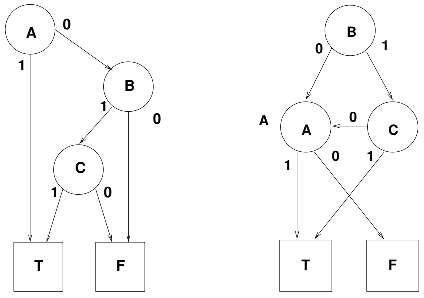
While our method of providing feedback is generally applicable to security systems, we focus on its use in ubiquitous computing environments. Good security feedback is particularly important for these systems for a variety of reasons:

Ubiquitous computing is an area of active research [RHC$^+$02; JFW02a] and good security feedback will help direct secure application design. Ubiquitous computing environments will be used mainly by non-technical users, where feedback is important for usability, since users either disable or work around obstructive security mechanisms. Access control in these systems can be more confusing to users than traditional distributed systems due to the inherent dynamism effected by a context-sensitive environment. Since access control policies may now depend on context in addition to traditional credential or permission based systems, without adequate feedback, it can be difficult to tell whether access was denied due to a bug in the system or due to user permissions changing in response to non-obvious changes in the context.

We believe that these features make ubiquitous computing systems an ideal test-bed for *Know*. Furthermore, such an environment allows us to assume coordination between security and feedback mechanisms for policies of resources in the entire system.

Test-bed: The Active Spaces project at the University of Illinois takes an operating system approach to ubiquitous computing environments. Gaia [RHC$^+$02], a middleware "meta" operating system interacts with all the devices in the space and provides a uniform programming interface to application developers. Gaia provides infrastructure services such as naming and context that applications can use, as well as security services for authentication and access control. The Gaia access control system [SNC02] uses an extension of the role-based access control system [FK92b]. We permit policies written in propositional form, which may include contextual propositions. Thus, permissions available to a user at any point in time depend on a variety of factors other than the user's credentials, and good feedback about access control denials is very important.

Policy protection: UniPro [YW03] provides a scheme to model the protection of resources, including policies, in trust negotiation. It allows policies to be treated as resources in the system, and allows the specification of policies to protect them.



(a) Variable ordering: a, b, c

(b) Variable ordering b, c, a

Figure 6.1: Example OBDDs for $a \vee (b \wedge c)$

Know uses the UniPro notation for writing meta-policies that protect the policies to decide what feedback can be provided to a user.

Representation: Ordered binary decision diagrams (OBDDs) [Bry86] are a canonical-form representation for boolean formulas where two restrictions are placed on binary decision diagrams: the variables should appear in the same order on every path from the root to a terminal, and there should be no isomorphic subtrees or redundant vertices in the diagram. A binary decision diagram is a rooted directed acyclic graph with two types of vertices: terminal and nonterminal. Each nonterminal vertex v is labeled by a variable $var(v)$ and has two successors, $low(v)$ and $high(v)$. We call the edge connecting v to $low(v)$ the 0-edge of v (since it is the edge taken if $v = 0$) and the edge connecting v to $high(v)$ the 1-edge of v. A single formula may be represented by multiple different OBDDs based on the order that variables in the formula are tested; however, given a particular variable-ordering, the OBDD structure is fixed (canonical form for that variable-ordering). Figure 6.1 gives an example of two OBDDs that each represent the simple boolean formula $a \vee (b \wedge c)$. The first is the canonical-form OBDD for the variable-ordering a, b, c and the second is the canonical-form OBDD for the variable-ordering b, c, a. To test for satisfiability, we start at the root node and test whether the variable at the root is *true* or *false*. If it is *false*, we follow the 0-edge, and if *true*, the 1-edge, and repeat this process. Eventually we reach either the T-node or the F-node (also called the 1-node and 0-node, respectively). If we reach the T-node, then the given assignment satisfies the formula; if we reach the F-node, it does not. For example, applying the assignment $\langle a = false, b = true, c = false \rangle$ to either of the OBDDs in Figure 6.1 tells us that the formula is not satisfied. We use OBDDs because they are a compact and graphical representation of boolean formulas. This allows us to use cost functions and shortest path algorithms to find conditions of satisfiability that are of "least cost" to the user.

Know stores access control rules as OBDDs, and can efficiently search these OBDDs for paths that satisfy the rules. When access is denied, the OBDD can provide information about alternate paths that would allow access. *Know* provides information to the user about such paths as feedback. The number of nodes in an OBDD can be exponential in the size of the boolean expression, but there are several heuristics to find an ordering that reduces the size of the OBDD, and in practice, boolean functions usually have a compact OBDD representation. We discuss this in more detail in Section 6.5. As mentioned earlier, *Know* provides feedback only if it can be done with acceptable overhead. Examples of Gaia system policies are presented in Sections 6.3 and 6.4.

6.3 Architecture

We augment the Gaia access control mechanism with our feedback component (*Know*). The Gaia access control mechanism intercepts all requests for service and checks them against the system policy. If disallowed, the access request is forwarded to *Know*, which prepares a feedback message for the user. Such a message contains a list of alternative conditions under which access to the given service is permitted. Alternatives may not always be available, either due to policy or computational resource constraints. In this case, the standard "Access is denied" message is provided. Since Gaia is highly context-driven, the feedback may suggest changes in context. For example, if a printer is inaccessible due to the current context (e.g., a meeting in the room disallowing the use of noisy printers), feedback may be of the form, "If you return when there is no meeting, then you will have access to Printer X."

When providing feedback to a user, a system must not compromise sensitive components of the system policy. For example, feedback of the form, "If you are a Motorola or IBM employee, then you will have access to this room" reveals sensitive information that IBM and Motorola may be collaborating on a project. To protect such information, *Know* augments the policies with meta-policies. A meta-policy governs access to a policy, thereby treating policies as objects themselves. When determining feedback for a user Alice, *Know* first checks the meta-policy to see what parts of the policy Alice is allowed to read, and constructs feedback using only those parts. UniPro [YW03] provides a generalized framework to protect parts of the policy with a policy, which in turn can be protected by another policy, and so on. Here, in the interest of simplicity, we focus on policies and their meta-policies (and not multiple levels of meta-policies). Henceforth, our notation will resemble that of UniPro, since we use a subset of its functionality. We now provide

Policy:

$$\begin{array}{rcl} R & : & P \\ P & \leftrightarrow & P_1 \vee P_2 \\ P_1 & \leftrightarrow & \textsf{User.role} = \textsf{Professor} \\ & & \wedge\ \textsf{User.department} = CS \\ P_2 & \leftrightarrow & \textsf{User.role} = CIA \end{array}$$

Meta-Policy:

$$\begin{array}{rcl} P_1 & : & \textsf{User.department} = CS \\ P_2 & : & false \end{array}$$

Figure 6.2: Policy for Example 1

some concrete examples of access policies, and their associated meta-policies.

Example 1 The access policy of an electronic door lock might allow access only to Computer Science professors or members of the CIA. When a person is denied access to the room, feedback of the form, "If you are a CS professor or a member of the CIA, then you will have access to this room" is potentially dangerous. Collaboration between the Computer Science department and the CIA could be sensitive information. Outsiders may also glean intelligence information about where CIA members meet. Clearly, we may not want to reveal parts of this access rule. Feedback of the form, "If you are a professor in Computer Science, then you will have access to this room" may be acceptable. A meta-policy would control this flow of information to denied users. Formally, we can represent the policy and meta-policy as shown in Figure 6.2.

A policy definition includes two types of expressions. An expression of the form $O : P$ means that an object O is protected by policy P, where policies themselves can be objects (since policies may be protected by meta-policies). An expression of the form $P \leftrightarrow E$ means that the policy P is defined by expression E. Expressions can contain both atomic propositions (e.g., $\textsf{User.department} = CS$) and references to sub-policies (e.g., $P \leftrightarrow P_1 \vee P_2$, where P_1 and P_2 are defined subsequently). The access policy for the room is $R : P$, which means that access to the room R is protected by policy P. P is defined as the disjunction of policies P_1 and P_2. P_1 is the policy, "User must be a professor in Computer Science." P_2 is the policy, "User must be a member of the CIA." Hence the policy P to access the room is "User must be a professor in Computer Science or the user must be a member of the CIA." Figure 6.3(a) shows the associated OBDD for this policy. The meta-policy $P_1 :$ $\textsf{User.department} = CS$ indicates that the policy P_1 may be revealed only to subjects in the Computer Science department, while the meta-policy $P_2 : false$ does not reveal P_2 under any circumstances[1]. For example, a denied student in Computer Science would receive the feedback "If you are a professor in Computer Science, then you will have access to this room," while a student in Civil Engineering will be informed, "Access is denied." In either case, no policy information involving the CIA is revealed. We discuss how to apply meta-policies to OBDDs through cost functions in Section 6.3.1.

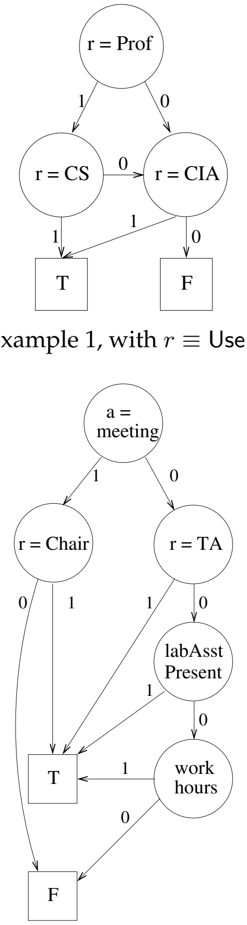
We make two assumptions here. First, we assume that any logical dependencies between atomic propositions are captured within the policy. For example, a policy may contain atomic propositions User.isAdult and User.isMinor. We know that $\mathsf{User.isAdult} \Leftrightarrow \neg\mathsf{User.isMinor}$, and hence feedback of the form "If you are an adult and a minor, you will have access" would be absurd. Such inconsistencies are avoided by either replacing occurrences of User.isMinor by $\neg\mathsf{User.isAdult}$ or by adding the logical rule $\mathsf{User.isAdult} \Leftrightarrow \neg\mathsf{User.isMinor}$ to the policy. This will avoid any inconsistencies in feedback. The second assumption we make is that all references to an atomic proposition a are protected by the same meta-policy. We elaborate on this in Section 6.3.2.

It is important to note that in all our examples we are careful to provide feedback as "If…then" clauses. This is important for policy protection. Feedback of the form, "Only professors may access this room" gives more information than "If you are a professor, then you will have access to this room." If both types of feedback were allowed, the user may infer from the latter feedback that there is a protected policy not being revealed. Hence, if feedback is consistent in its use of "If…then" clauses, users will not gain any extra information about protected policies.

Example 2 Since we are interested in providing useful feedback for complex policies, we provide a more complex example in Figure 6.4. We first present the policy P for a printer A, and then augment it with a meta-policy.

To understand this policy, first note that printer A is noisy, and so, disruptive to meetings being held in the room. During meetings, printer access is restricted to the person in charge of the meeting. P_2 and P_6 ensure that nobody will disturb the meeting by using the printer, but the meeting chair may use the printer if needed. When there is no meeting in effect, we would like to grant access to the printer to anybody during normal business hours (P_1, P_5). At other times, users are only allowed to access this printer while no meeting is in progress, and in the presence of a Lab Assistant (P_1, P_4). Teaching Assistants have 24-hour access (P_1, P_3) to the

[1] In our examples we omit meta-policies of the form $P : true$ for clarity. In practice however, all meta-policies may be assumed to be of the form $P : false$ unless a meta-policy is explicitly specified.



(a) Example 1, with $r \equiv$ User.role

(b) Example 2, with $r \equiv$ User.role, $a \equiv$ Context.Activity, $t \equiv$ Context.time

Figure 6.3: OBDDs for the examples

.

Policy:

$$\begin{array}{rcl}
A & : & P \\
P & \leftrightarrow & P_1 \vee P_2 \\
P_1 & \leftrightarrow & \textsf{Context.activity} \neq \textsf{meeting} \wedge (P_3 \vee P_4 \vee P_5) \\
P_2 & \leftrightarrow & \textsf{Context.activity} = \textsf{meeting} \wedge P_6 \\
P_3 & \leftrightarrow & \textsf{User.role} = \textsf{TeachingAssistant} \\
P_4 & \leftrightarrow & \textsf{Context.labAssistantPresent} = true \\
P_5 & \leftrightarrow & \textsf{Context.workingHours} = true \\
P_6 & \leftrightarrow & \textsf{User.spaceRole} = \textsf{MeetingChair}
\end{array}$$

Meta-Policy:

$$\begin{array}{rcl}
P_3 & : & false \\
P_6 & : & \textsf{User} \in \textsf{Context.activityMembers}
\end{array}$$

Figure 6.4: Policy for Example 2

printer to perform their duties (again, as long as there is no ongoing meeting in the room). Figure 6.3(b) shows the associated OBDD for this policy.

Consider the case when a Student is denied access to the printer. An "Access is denied" message may be confusing to the Student who was able to access the printer the previous day. In the spirit of offering the user a consistent view of the system's policies, we would like to inform the user why the access was denied. Was it because a meeting was in effect? Should the user come back during regular business hours when there is no meeting? Should the user be informed that there is no lab-assistant in the room and that it is past business hours? Clearly there are several useful options available to the user. Now consider some other options that may not be of much help to the user. Let us say that access was denied outside of working hours, and no meeting was taking place in the room. Feedback of the form "Access is denied, but if you are a Teaching or Lab Assistant, then you will have access to this printer" is clearly less useful to the user, since becoming a Teaching or Lab Assistant is a non-trivial task.

Consider a final scenario when a person is denied access to the printer because of an ongoing meeting. If the user is not a member of the meeting, feedback of the form "If you are the meeting chair, then you will have access to this printer" may suggest to users that they request the chair to print documents. This is clearly disruptive, so we would like to disclose this fact only to people who are participating in the meeting. Consider the meta-policy for the policy provided above, shown in Figure 6.4. Using this meta-policy, we restrict feedback provided to users.

Users who are denied access are not informed that Teaching Assistants have 24-hour access ($P_3 : false$), since this may result in several students accosting their Teaching Assistants for their personal printing needs. Further, users who are denied access to the printer during an ongoing meeting, are only informed about the meeting chair's printing capability if they are a member of the current meeting (P_6 : $\mathsf{User} \in \mathsf{Context.activityMembers}$).

We have presented examples of feedback that a user may, or may not, find useful. Furthermore, there were some examples of feedback that were restricted or eliminated by the meta-policy due to privacy concerns. Providing useful feedback in the face of restrictions by a meta-policy, and the relative usefulness of the various options available to the user, suggests the use of a *cost function*. This cost function can evaluate each feedback option, and generate an ordering that says one option is better than another. For example, the system may provide the user with the three most useful options as determined by the cost function. We now describe cost functions in more detail and formalize the notion of "feedback."

6.3.1 Cost functions

When the access policy for a resource is not satisfied, we can try to compute all the paths from the root of the OBDD of the policy to the *true* node. Essentially, a set of such paths will be presented as feedback to the user since they represent assignments that satisfy the policy. Some of the edges followed in the path will correspond to conditions that do not currently hold. Consider Example 2, with a meeting in progress, and a TA trying to access the printer. One possible path from the root to *true* is $\mathsf{Context.activity} \neq \mathsf{meeting}, r = TA$. This requires one change since there is a meeting in progress. Other paths may require more changes. The number of changes that must be made for each path, and the relative difficulty in making certain changes over others, suggests the use of cost functions to rank the options. We first introduce some notation and a formal definition of *feedback*.

We use the notation $S \models P$ to indicate that a policy P is satisfied under the atomic propositions specified by S. For example, we could have $S = \{\mathsf{Context.meeting} = false, \mathsf{Context.workingHours} = true\}$. In the notation $S[a] \models P$, $S[a]$ is the set of atomic propositions in S along with any update provided by a. In our example, $S[\mathsf{Context.meeting} = true]$ = $\{\mathsf{Context.meeting} = true$, $\mathsf{Context.workingHours} = true\}$. This notation naturally extends to a set of updates, e.g., $S[A]$, where A is a set of atomic propositions. Let C be the set of atomic propositions relating to the context of the system and U be the set of atomic propositions specific to the user (identity, role, etc.). Given a policy P and a user U, the user is granted access when

$C \cup U \models P$, and denied access when $C \cup U \not\models P$. In essence, if $C \cup U \not\models P$, then a set of updates X such that $(C \cup U)[X] \models P$ constitutes a feedback option to the user.

To formalize the notion of feedback, let $\Pi = \{\pi_1, \ldots, \pi_n\}$ be the set of paths from the root node to the *true* node in the OBDD of P. Let π'_i be the set of atomic propositions that appear in $\pi_i \in \Pi$ and whose truth values differ in $C \cup U$, i.e., the set of propositions that must be changed (or a set of updates to the state) for the policy to be satisfied. Let $\mathcal{F} = \{\pi'_1, \ldots, \pi'_n\}$. Note that $(C \cup U)[\pi'_i] \models P$ for all $\pi'_i \in \mathcal{F}$. We define any subset F of $\mathcal{F}$ to be the $feedback$ offered to the user. In other words, each *feedback option* f_i in the *feedback* F corresponds to a set of atomic propositions the user must change to be granted access. $\mathcal{F}$ is the set of all possible feedback options available to the user. Since $\mathcal{F}$ can be very large, our primary goal is to find a way to offer the user only a few relevant feedback options in $\mathcal{F}$. We do this through the use of cost functions. The cost function assigns a cost to each $f \in \mathcal{F}$, and returns the k lowest-cost feedback options, where k is a tunable parameter.

A naïve cost function could assign the same cost to each change, in which case the user would be given feedback with the least number of changes that need to be made to access a resource. For example, we could sort the elements f_i of $\mathcal{F}$ in ascending order of $|f_i|$ (number of atomic propositions in f_i) and return the first k choices. However, changing roles might be more difficult than changing context. For example, a Student may be able to come back at a later time, but it would be extremely difficult to acquire a Professor role. This suggests the use of more sophisticated cost functions.

We need to define an appropriate cost function that is applied to edges in the OBDD as edge weights. Using these weights we can use shortest path algorithms from the root to *true* to provide feedback with lowest total cost. Running Dijkstra's algorithm gives us a path with lowest total cost in polynomial time. There are several proposed algorithms for k shortest paths for graphs. Eppstein [Epp94] presents an algorithm that computes k shortest paths in time $O(m + n\log n + k)$, where n is the number of vertices, and m is the number of edges in the graph. This is the best known bound for k shortest paths in directed acyclic graphs. Since an OBDD with n nodes has $2n - 4$ edges (two children for each node, except the *true* and *false* nodes), the complexity for computing the k shortest paths in an OBDD is $O(n\log n + k)$.

Let A be the set of atomic propositions in the policy P. We define a cost function $c : A \times \{0, 1\} \rightarrow \mathbb{R}^+ \cup \{\infty\}$, where $\mathbb{R}^+$ is the set of non-negative real numbers. This function tells us the cost to change an atomic proposition, an in effect, the cost to follow a 0-edge or a 1-edge for a node in the OBDD. An infinite cost disallows any changes to the current value of the proposition. When a request for access is denied, let $T \subseteq A$ be the set of propositions that evaluate to *true*, and $F \subseteq A$ be the set of those that evaluate to *false*. We define $c(t, 1) = 0$ for $t \in T$ and $c(f, 0) = 0$ for $f \in F$ since there is no cost to maintain atomic propositions that are satisfied under the current conditions ($C \cup U$), and we would like to assign non-zero cost when a user must *change* some atomic proposition. Cost functions will differ according to their assignments to $c(t, 0)$ for all $t \in T$ and $c(f, 1)$ for all $f \in F$. Now, for all $a \in A$, assign the weight $c(a, 0)$ to the 0-edge of a, and $c(a, 1)$ to the 1-edge of a. What results is a directed acyclic graph with weights assigned to each edge. We can now apply k-shortest path algorithms to this graph to get the k lowest-cost paths, which correspond to the k lowest-cost feedback options. For small k, the running time for such algorithms is dominated by the structure of the OBDD and not k. Specifically, since we expect to have $k < n$ (for example $k = 3$ might be sufficient), the running time is $O(n \log n)$. Our naïve cost function that considers all changes to be equally expensive would set $c(t, 0) = 1$ for all $t \in T$ and $c(f, 1) = 1$ for all $f \in F$. Hence the total cost of any path is equal to the number of propositions that need to be changed under the given conditions.

6.3.2 Meta-policies

Now that we have described the basic algorithm for computing feedback using OBDDs and shortest path algorithms, we must modify the algorithm to honor the meta-policies. Each meta-policy determines whether a user can read certain nodes in the policy's OBDD. Let $D \subset A$ be the set of nodes forbidden by the meta-policy. For each $d \in D$, we assign infinite cost to the edge that effects a change in the current value of d. This does two things: first, it prevents shortest path algorithms from exploring a change in d and hence does not return any feedback options that require a change in d. Second, since this proposition d cannot be changed, it will not appear within a feedback option, which includes only those propositions that must be changed. Since no atomic proposition that is precluded by the meta-policy appears in any feedback option, *the feedback given to the user honors the meta-policy*. We assume that all nodes corresponding to a particular atomic proposition a are protected by the same meta-policy, allowing us to perform such a transformation. Finding efficient ways of computing consistent feedback where references to the same atomic proposition are protected by different meta-policies is left to future work.

6.3.3 A useful cost function

We now present a useful cost function that improves on the results of the naïve cost function in the context of ubiquitous computing environments. The useful cost function forbids the exploration of certain, obviously undesirable, options. Paths to the *true* nodes will still be graded according to the number of changes, but certain changes are forbidden by the cost function, and, therefore, not considered.

Activities: Gaia policies for a smart space depend on the current activity in that space. Example 2 showed the policy for the activities *meeting* and *no meeting*. Consider a space with the possible activities of *meeting, conference, reception*, *presentation* and *no activity*. If a user is denied access to a resource during a *meeting*, feedback of the form "During a *conference*, if you are the Chair, you will have access" is not very useful. So we only provide the user feedback for the current activity, and the absence of any real activities (*no activity*). Hence, we apply an infinite cost to all 1-edges for the activities that are not current (the 0-edges will have 0 cost) and apply a cost of 1 to the 0-edge for the current activity and the 1-edge for *no activity*. This will cause *Know* not to explore feedback for other activities, but do so for both, the current activity and no activity.

Roles: This cost function assumes that it is difficult for a user to obtain a new role. Let N be the set of nodes in the OBDD that tests for a role that the user has not activated. For each node in N, assign infinite cost to the 1-edges (the 0-edges will have 0 cost). Hence the system will not provide any feedback that requires a user to obtain a new role. A user may choose to activate only certain roles in the system. If the user is not satisfied with the feedback options obtained, he or she may activate another role and get better feedback.

Meta-policy: As described in Section 6.3.2, we assign infinite costs to all edges that require a change to variable assignments forbidden by the meta-policy.

For a given feedback option (path from root to the *true*-node in the OBDD), the cost is the number of changes to be made. Since certain edges are forbidden due to infinite cost, feedback provided to the user will not require any changes in the current role, and will only be for the current activity, or no activity. We believe that this cost function is useful in the context of current policies and activities in Gaia. In our analysis, we provide a detailed example policy and show how this cost function provides more useful feedback to the user than the uniform cost function.

6.4 Implementation

We have built a prototype of the *Know* system, which will be incrementally deployed within Gaia. In this section we describe the implementation and results from a preliminary evaluation. We present results of *Know* running with an example access control policy for videoconferencing equipment located in a kiosk within a multi-purpose business center.

The system access policy is represented as an OBDD, which is then transformed into a weighted graph that is specific to access requests. An appropriate cost function, along with the system meta-policy, is used to assign weights to the edges. Finding the k shortest paths to the 1-node of the OBDD gives us k sets of assignments to the variables that will satisfy the access control rules, and thus, describe k situations under which the particular operation is allowed.

The first step is to generate an OBDD from the system access control policy. We use the BuDDy [LN99] library, which uses heuristics for optimizing the generated OBDDs. The end result of this is an OBDD that represents all allowable ways to perform a particular action (or access a particular resource). If the requested action is permitted, *Know* is not needed. If not, *Know* attempts to find alternative paths in the OBDD that would permit the operation, *i.e.*, paths in the OBDD from the root to the 1-node.

Alternative paths are found by using the Eppstein [Epp94; Gra] algorithm to find the k shortest paths from the root to the 1-node in this OBDD. Weights are assigned to the edges of the OBDD graph based on the cost function and the current values of the user roles and context variables. Selecting a suitable cost function is site-specific—the weights assigned to the different changes will depend on the nature of tasks that are normally performed by users of the system. We provide results from the two cost functions described earlier—the naïve cost function (which counts the number of changes required) and the "useful" cost function (which treats role changes as more difficult to achieve than context changes).

Know then outputs the necessary changes that must occur to satisfy the alternative paths. It is up to the user to choose between these suggestions, and to retry the request after following the suggestion.

6.4.1 Evaluation

We illustrate this entire process with its application to a sample policy that governs the access to videoconferencing devices in the business center of a hotel. In addition to computers, the business center also contains devices such as printers, fax machines, cameras for videoconferencing and so on. The business center is located in the conference hall, and hotel guests and other members who have signed up are normally allowed to use the devices as per the security policy. The conference hall is also rented out for activities such as meetings, conferences, or receptions, during which time use is restricted to participants of this activity, as per the policy configuration by the organizers. Users present their credentials to enter the business center, in the form of a smartcard (a conference badge or a hotel room key) and the system uses this information to restrict access and provide useful feedback. We present here the rules that affect access control to the camera for the videoconferencing system.

The basic policy is as follows:

- When no activity is scheduled for the room, supervisors, hotel guests or other registered users can use the videoconferencing equipment during the business day. Visitors are also allowed to use the facilities if an operator is present. Hotel guests may also use the system during non-business hours, but others may not.

- When an activity (such as a videoconference) is scheduled, only registered activity participants and supervisors are allowed to use the system.

- Use of the videocamera is disallowed for regular participants if the videoconferencing activity being undertaken in the conference center is labeled as confidential. However, the meeting supervisor may still turn on the videocamera if all participants have the required security clearance.

- Maintenance activities are performed by designated personnel.

- Finally ambient temperature above 30C indicates some problem with the air-conditioning/cooling system, and camera use is prohibited until temperature reaches the allowed range. Similarly, overcrowding the room will violate the fire safety codes and cause access to the camera to be denied.

The meta-policy that governs feedback contains the following rules:

- Information about confidential activities is only provided to the meeting supervisor. Thus an unauthorized user trying to access the videocamera during a confidential activity will not be informed that a confidential activity is going on, but just that access is denied at that time. Similarly, feedback about the presence of uncleared users is only given to the meeting supervisor.

- Information about maintenance activities is not provided to other users.

Policy:

$$
\begin{array}{rcl}
C & : & P \\
P & \leftrightarrow & P_1 \vee \ldots \vee P_{10} \\
\ldots & & \\
VC & \leftrightarrow & \mathsf{activity} = \mathsf{VideoConference} \\
 & & \wedge\neg(A_1 \vee \ldots \vee A_n) \\
CA & \leftrightarrow & \mathsf{Context.isConfidential} = true \\
RS & \leftrightarrow & \mathsf{User.role} = \mathsf{Supervisor} \\
NU & \leftrightarrow & \mathsf{Context.UnclearedUsersPresent} = false \\
NH & \leftrightarrow & \mathsf{Context.cameraOverheated} = false \\
NF & \leftrightarrow & \mathsf{Context.roomFull} = false \\
RP & \leftrightarrow & \mathsf{User.role} = \mathsf{Participant} \\
P_7 & \leftrightarrow & VC \wedge CA \wedge RS \wedge NU \wedge NH \wedge NF \\
P_8 & \leftrightarrow & VC \wedge \neg CA \wedge (RP \vee RS) \wedge NH \wedge NF
\end{array}
$$

Meta-Policy:

$$
\begin{array}{rcl}
\ldots & & \\
CA & : & \mathsf{User.role} = \mathsf{Supervisor}
\end{array}
$$

Figure 6.5: Example policy used for evaluation

The access control rules for this policy above are presented in Figure 6.4.1. In our implementation, access to the camera C is protected by policy P. Policies $P_1, \ldots, P_{10}$ describe the various rules presented above, where P_7 and P_8 are rules pertaining to the VideoConference activity. In the interest of brevity, we only present the rules relevant to the VideoConference activity in Figure 6.5. This policy states that during a confidential VideoConference, only a Supervisor can access the camera as long as there are no uncleared users present. During a non-confidential VideoConference, any Participant or Supervisor can access the camera. The room must never be Overheated or Full during a VideoConference. The second rule in the meta-policy states that only Supervisors will be made aware of Confidential activities (or the lack thereof). Hence if an ordinary user is denied access to a camera, the user will not be told that there is a confidential conference in progress (this information itself is deemed sensitive). Since there can be only one activity at any given time, the policy specifies $VC \leftrightarrow \mathsf{VideoConference} \wedge \neg(A_1 \vee \ldots \vee A_n)$, where $A_1, \ldots, A_n$ are the remaining activities.

The OBDD generated by the above policy has 17 variables and 35 nodes (in contrast, a binary decision tree would have at least 2^{17} nodes).

To evaluate *Know*, we try to access the videocamera under a variety of situations, and present the suggestions provided by *Know* using each of the two cost functions described earlier, which we designate as the “naïve” and the “useful” cost function. Since the useful cost function just restricts information about role and activity change, feedback from the useful cost function will just be a (more useful) subset of the feedback from the naïve cost function. We describe some of the experiments below for $k = 4$. The run-time overhead for *Know* to find these suggestions was negligible—in the order of milliseconds. Since OBDDs are just a representation of the access control policy, they can be constructed ahead of time and only need to be re-computed if the policy changes. Assigning weights to the edges of the OBDD is performed each time a request arrives, since the weights depend on the current values of the context variables and user credentials. Since *Know* runs only when access is denied, it has no performance overhead on successful requests. We now describe the situations and results in detail:

- A Visitor tries to use the camera during business hours, but no Operator is present. There is no activity in session. With the naïve cost function, *Know* suggests that the user come back a) as a HotelGuest b) as a RegisteredRoomUser c) when an Operator is present, or d) as a Supervisor. The useful cost function suppresses the suggestions involving a role change, and only advises the user to come back when an Operator is present. This simple example illustrates the basic functionality of *Know*.

- If a HotelGuest tries to use the equipment during working hours when the room is too hot and there is no activity in session, *Know* correctly suggests that the user try again a) when room is not Overheated b) when room is not Overheated and it is out of business hours and c) when room is not Overheated and as a RegisteredRoomUser instead of a HotelGuest, or d) when room is not Overheated, as a Supervisor, instead of a HotelGuest. The useful cost function only offers the first two suggestions because it does not recommend role changes. Clearly, the only change *required* is for the temperature to be reduced, but *Know* does not presently restrict suggestions that are subsets of others. This may be useful in some situations.

  Maintenance operations are allowed even in overheated conditions, and a straightforward search through the policy might have offered the suggestion to try coming back as a MaintenanceWorker. However, the system metapolicy forbids the disclosure of information about maintenance permissions, so this option is correctly ignored by *Know*.

- During a Confidential videoconferencing activity and regular working hours, if a Participant tries to access the videocamera when users without the required security clearance are present, the naïve cost function suggests the user come back a) as a HotelGuest when no activity is in progress, b) as a RoomUser when no activity is in progress, c) as a Visitor when no activity is in progress, or d) as a Supervisor when no activity is in progress. The useful cost function does not offer any feedback, because there is no useful option for the Participant.

  One possible suggestion is to inform the user that this operation is not permitted during a confidential activity and to suggest re-trying when no confidential activity is being undertaken, but the system meta-policy precludes any information about confidential activities from being revealed, so this suggestion is not offered. Note that it is possible for users to correlate feedback from different sessions to infer the *existence* of hidden atomic propositions. For example, the presence or absence of a confidential activity results in differing feedback. Care must be taken while writing the policies and meta-policies to prevent the leakage of the *identity* (e.g., confidential activity) of the atomic proposition.

- If a Supervisor tries to use the camera when the room is reserved for a confidential VideoConference and uncleared users are present, the uniform cost function suggests that the user come back a) after changing the activity type to be non-confidential b) when no uncleared users are present, c) when there is no activity scheduled, or d) as a Participant after changing the activity type to be non-confidential. The useful cost function suggests the first three options. Note how the Supervisor is given feedback regarding Confidential activities, as opposed to a Participant in the previous scenario.

While the above examples are fairly simple, they validate our hypothesis that *Know* can provide useful information about alternatives when access is denied, that it can do so without compromising privacy or confidentiality requirements of the security policies, and that this can be achieved with negligible performance overheads. We are in the process of integrating *Know* fully with the Gaia system, after which we can perform larger-scale studies.

6.5 Discussion

Usability has been recognized as an important concern for security systems since the early days [SS75] of research in computer protection systems; however, in practice, usability issues have not been a primary consideration for security designers. Usability concerns are especially important in ubiquitous computing environments, since the objective of ubiquitous computing is to blend into the background and allow the user to perform his or her tasks without having to pay attention to the computing environment. Work on the human-computer interface aspects of security [Yee02; ZS96] have identified consistent feedback as an important aspect of usability. We posit that providing useful feedback about access control decisions is a step in the right direction. Users can then obtain a better picture of the security policies and can access resources accordingly.

While policy feedback improves system usability, one of the major concerns is information leakage. Meta-policies allow system administrators to treat the policy just like any other system resource, and configure access to it accordingly. Thus, providing feedback does not leak any unauthorized information. We recognize that writing effective meta-policies that are robust against statistical inferencing remains a challenge, and further research is indicated.

Computing feedback options is equivalent to computing variable assignments to satisfy a boolean formula. In general, computing assignments for satisfiability (SAT) of a boolean formula is NP-complete, and finding least-cost assignments (Weighted SAT) is an NP-hard optimization problem [OM87]. Therefore, we cannot expect to compute "least cost" feedback in all cases. We propose that feedback should be provided if it can be done with acceptable computational and storage overhead. Responses can be cached to improve efficiency, especially in situations where users might make repeated requests for a resource.

OBDDs are an efficient and compact representation of boolean formulas, and tests for satisfiability are efficient. It is well-known that the size of OBDDs depends on the variable ordering (the order in which variables are tested in an OBDD), and in certain cases it is not possible to reduce the exponential state space of decision trees. In fact, determining a suitable variable ordering (yielding a minimum-sized OBDD) has been shown to be NP-complete [BW96]. Given these challenges, it is comforting to know that commonly encountered functions have reasonably sized OBDDs and there are several heuristics (e.g., group-sifting [PS95] is one of the popular methods, also see [BRKM91; FMK91]) to determine variable orderings for adequately small (non-exponentially sized) OBDDs. Degenerate cases usually involve functions that behave differently for all possible assignments (e.g., output of an integer multiplier [Bry86] and integer division [HY97]), and we do not expect such state space explosion in the case of our policies. Furthermore, efficient heuristic algorithms such as A^* [Kor02] may improve search performance. While we chose OBDDs for the convenience of using graph algorithms, other representations of policies may also prove useful. If policies are represented in Disjunctive Normal Form (DNF), feedback computation can be performed in at most $O(U \log U)$ where

U is the number of clauses in the DNF representation. However, converting policies to DNF may result in large U. We are currently studying the relative merits of such alternative representations.

The search space could be reduced by allowing the user to specify constraints such as the maximum number of changes v that he or she is willing to accept. In that case, even a naïve brute-force algorithm for computing feedback would take time $O(n^v)$, which might be acceptable overhead for small v.

An interesting avenue for future work is the study of suitable cost functions. More nuanced cost functions could better reflect the relative difficulty of changing propositions. Selection of an appropriate cost function will be influenced by various factors, such as user preferences and system usage patterns. Another option could be to allow users to specify their own cost function to tailor the *Know* feedback. Learning algorithms [ZK03] could be used to improve feedback over time by allowing users to rate the feedback received. For example, the learning algorithm can use these examples to update the costs for atomic propositions and eventually "learn" the user's preferences. Chajewska et al. [CKP00] explore approaches for making decisions based on imperfect utility functions. The system tries to learn a user's utility function through a process called *utility elicitation* and computes the optimal decision based on the current utility function.

Other possible approaches include Knowledge bases [LL01] and planning algorithms [LaV06] for storing access policies, and rules to express meta-policies and user preferences. Planning algorithms can be used to compute the least cost number of predicates that can be changed to gain access to a resource. The main advantage of this approach would be the use of additional inference rules for better feedback. For example, if Alice is a student in a particular course, it is unlikely that she is also the TA for that course. More complex approaches such as knowledge bases and the use of non-monotonic logics such as "common sense reasoning" [LGP$^+$90] can improve the quality of feedback, although such approaches can be computationally intensive [ET01]. Our current solution for reasoning applies OBDDs and cost functions that hide the complexity of the general logical inferencing problem and provides an efficient solution to our specific application of computing feedback for access control policies.

McGuinness and Silva [MdS04] propose an Inference Web (IW) that aims to provide explanations for answers to queries based on assertions from diverse sources. Yes or no answers to questions like "Is red wine good with fish?" are seldom useful to users if the conclusion is not justified or suitably explained. Moreover, facts or rules used in the explanation may not come from trustworthy sources. The authors address "knowledge provenance" or the origins of asserted facts, and how this can be included in the explanations. For example, a user may trust conclusions regarding wine if the supporting facts come from the Wine Spectator magazine more than conclusions based on facts from Sports Illustrated. IW is also meant to be a "proof abstractor" that seeks to provide concise explanations of lengthy proofs using rewrite rules and proof annotations. In this context, *Know* is a system that provides "explanations" to users as to how they can gain access to a denied resource, without concerns about knowledge provenance. In terms of proof abstraction, *Know* provides a mechanism to provide the most relevant explanations to users by using cost functions, and informs users about propositions that need to be changed, resulting in compact feedback. Unlike *Know*, IW does not explicitly address the confidentiality of facts used in explanations. Meta-policies such as those in *Know* may be used to restrict what explanations are given to particular users where certain explanations may be deemed as sensitive information, while the conclusions are not. As mentioned for knowledge bases, the IW approach to providing feedback can be used in conjunction with inference rules that can reason about useful feedback. When *Know* gives the user feedback, the system can attempt to justify why the feedback is useful to the user. For example, a user with similar attributes may have asserted its usefulness.

Another question of interest is the feedback mechanism, since ubiquitous computing environments use a variety of mechanisms to interact with users, and text messages on a display monitor may not always be available or appropriate for system feedback. Audible feedback may reveal one user's feedback to other nearby users, which may also not be appropriate. Hence *Know* might take into consideration the credentials of other users present in the room for restricting feedback based on the delivery mechanism.

6.6 Summary

We have presented *Know*, a system for providing feedback to users about access control policy decisions. When the system denies a user access to a resource, *Know* suggests useful alternatives for the user to gain access. While a list of all possible alternatives is likely to be large and not very useful, *Know* restricts the options presented to the user to a smaller set of useful options by the use of appropriate cost functions. An important consideration is that this process should not leak any information that would compromise the confidentiality of access policies. This is achieved by using a meta-policy to represent the required protection for the policies themselves. We presented qualitative performance results from a prototype implementation.

Lastly, we believe that research in the area of providing useful feedback to denied users has not been adequately researched, and to the best of our knowledge this is the first attempt at integrating useful policy feedback with policy protection.

7 Conclusions

We have presented models for achieving privacy in ubiquitous computing environments. We addressed the problems of communication privacy, unlinkability of users' accesses to services, and policy privacy during access control feedback. We now present our conclusions, summary of contributions, and indicate future research directions.

7.1 Conclusions

We developed a model for trustworthy and anonymous routing, where users can influence their paths of communication based on qualitative security properties of routers and quantitative representations of trust based on perceived threat. Our main contribution was the novel application of model checking constructs, Constrained Linear Temporal Logic, and shortest path algorithms for finding paths with a high quality of protection. We argued that privacy protocols that assume independence are not adequate in organizational settings, where trust relationships can no longer be ignored. Furthermore, the discretionary privacy demands of users had not been adequately researched in networking environments, where security focused on the mandatory security requirements of the organization.

We showed how a concerned user can present a set of audit flows to a decision engine, which analyzes the potential unlinkability threats of the user's access transactions across these flows [KNC05]. The user can then negotiate a set of policy constraints with the system to provide unlinkability of audit flows with respect to specific perceived threats. We applied graph theoretical techniques to generate policy constraints for audit records, and proved how our system is secure and precise under strong tranquility. We also proposed an approach based on versioning that maintains security by trading off precision for evolving protection state.

We addressed the issue of providing users with useful feedback about access control decisions while maintaining the privacy of system policies. We presented a security feedback mechanism called *Know* [KSC04], which addresses privacy properties such as the confidentiality of policy authorizations, and showed how sensitive permissions in the system can be protected while providing users with useful feedback on access control decisions. Relevant feedback enhances system usability, especially in systems where permissions change in unpredictable ways depending on contextual information. However, providing feedback indiscriminately can violate the confidentiality of system policy and the privacy of other users. *Know* achieves a balance between system usability and the protection of security policies by allowing administrators to specify "meta-policies" for regulating feedback based on the system's perceived threats. Our solution applied ordered binary decision diagrams, cost functions, and shortest path algorithms for computing useful feedback while providing policy protection. *Know* is the first model to address the need of useful access control feedback while honoring the privacy and confidentiality requirements of a system's security policies.

7.2 Summary of contributions

We now address the evaluation criteria in Section 3.3.

1. **Trustworthy routing**

   (a) In Section 4.5.3 we showed how the network can be represented as a state transition diagram with a labeling function to represent the properties (or attributes) of routers and links. This representation facilitates the use of linear temporal logics for specifying path properties based on trust relationships in the network.

   (b) In Section 4.6.5 we presented a policy language based on Constrained Linear Temporal Logic for specifying privacy properties for routes in the network based on a user's perceived threat to communication privacy. This policy language captures qualitative and quantitative attributes, and allows the user to specify constraints that include mathematical relations on boolean and real-valued variables.

   (c) In Section 4.6.6 we showed how simple and efficient graph transformations can be made to satisfy the policy language. Specifically, *any* route computed from the transformed graph will satisfy the specified policy. This property is useful while applying quantitative trust models, which rely on optimizing paths on a weighted version of the modified graph.

   (d) In Section 4.7 we presented a *combiner* function for computing the trustworthiness of paths. This combiner function is general enough to capture a host of semantic notions of trust.

   (e) In Sections 4.7 and 4.8 we evaluated and identified several feasible and infeasible trust models. Several NP-hardness results were presented, showing that these trust models are hard to use in practice.

   (f) We showed how our expressive policy language, combined with the use of feasible trust models, can efficiently yield paths of "high confidence" using shortest path algorithms and other techniques.

2. **Unlinkability**

   (a) In Section 5.3.2 we showed how the flow of audit flow information can be captured with a graph representation.

   (b) In Section 5.3.3 we showed how conflicts, or potential threats to linkability, can be identified efficiently.

   (c) In Section 5.3.4 we showed how users can specify discretionary unlinkability policies based on identified conflicts based on perceived threat to unlinkability.

   (d) In Section 5.3.5 we showed how polices are attached to audit information and enforcement of these policies is decentralized and efficient.

   (e) In Section 5.3.5 we also proved the *security* and *precision* properties of our approach. We showed how security can be maintained by trading off precision for evolving protection state by using versioning.

3. **Feedback and policy protection**

   (a) In Section 6.3 we showed how administrators can authorize various levels of feedback based on the user's credentials. This was done using meta-policies to encode the perceived threat to policy confidentiality.

   (b) In Section 6.3.1 we showed how cost functions can be employed to rate feedback based on its usefulness to the user.

   (c) In Section 6.3 we showed how *Know* represents policies as Ordered Binary Decision Diagrams (OBDDs). These OBDDs are a graphical representation of the policy. We showed how the OBDD of a policy is modified based on the meta-policy and cost-function. Shortest path algorithms can be used to compute feedback from the transformed OBDD.

   (d) *Know* allows users in ubiquitous computing environments to get authorized feedback on access control decisions, giving them a clear picture of their access rights in the system. This approach makes ubiquitous computing environments more usable and more secure, since frustrated users could otherwise disable security features that are confusing.

7.3 Future Research

We now indicate directions for future work in trustworthy routing, unlinkability of access transactions, and policy feedback.

In Chapter 4 we discussed several semantic models of trust. In Section 4.7.2 we showed how multiplicative combiners can be used for probabilistic models of trust. For example, the confidence value of 0.9 for the attribute "Physically Secure" can mean that the probability that the machine is physically secure is 90%. We showed how the overall confidence of the path, the probability that the entire path is physically secure, is simply the product of these probabilities, assuming they are independent. In modeling, one attempts to abstract or approximate the real probability distribution with one that is tractable. For example, Markov models in queueing theory assume a memoryless distribution for arrivals, which greatly simplifies the mathematics. While it is certainly true that security failures in a network are not independent, we would like to examine the space of probability models that are more powerful than the independent probability model, but still computationally efficient to compute. In particular, we would like to avoid approaches with time or space complexity that is exponential in the size of the network. We have been exploring the use of Markov Random Fields and Bayesian networks to represent correlated confidence values. While these models appear to be intractable in general, we would like to identify restrictions on these models that yield tractable solutions. For example, Dugan et al. [ST86] explore combinatorial approaches in conjunction with Markov models to constrain the state-space. Such approaches may yield useful and tractable models for trustworthy routing. We would also like to study more powerful policy specification languages for trustworthy routing. Ultimately, user studies are required to identify a usable policy language or a higher level policy specification tool, which would allow regular users to specify privacy policies in a more intuitive way. We also identified several semantic models of trust for which it is NP-hard to find paths of highest confidence. Further research is required for approximation techniques and heuristics for computing paths with reasonably high confidence even if the optimal solutions are elusive.

In Chapter 5 we presented an access control based approach for providing unlinkability to users in ubiquitous computing environments. We would like to study its use in combination with privacy routing protocols. For example, routers may log connection information for legal purposes. The user may want to restrict the linkability of this information by attaching policies to these records. Further research is needed to analyze the security properties of this approach. We would also like to combine this approach with quantitative models so that the user and system can negotiate unlinkability policies that maximize the "utility" of both parties. We presented an approach based on versioning to trade off precision for evolving protection state. It may be possible to improve precision by more complex policy analysis.

In Chapter 6 we presented a meta-policy language for administrators and a cost-function based approach for rating the usefulness of feedback. Further research is required in making meta-policy specification for administrators easier. *Know* assumes that the administrators have guidelines for setting meta-policies, but this is still not well understood. We believe it may be useful for administrators to specify "global" meta-policies that apply to all policies in the system. These global meta-policies can be applied uniformly to all policies in the system. This can benefit from RBAC by grouping objects and subjects into roles, and specifying meta-policies for which sets of subjects can obtain feedback about other sets of subjects for a particular set of objects or resources. We would also like to study other representations of policies besides OBDDs. Since the problem of computing useful feedback is NP-hard in general, it would be useful to combine the power of several representations. For example, *Know* could store disjunctive form policies if the OBDDs are too large. We have evaluated two cost functions that are easy to specify. More research is required to compute cost functions for different users. For example, the system can try to learn a user's preferences and adjust the cost functions appropriately. Furthermore, a user may want to supply his or her own cost function. *Know* provides a framework for such research by reducing the problem of usable feedback to computing useful cost functions.

References

[ABKM01] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proc. 18th ACM SOSP, Banff, Canada*, October 2001.

[AH92] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Real Time: Theory in Practice, Lecture Notes in Computer Science 600, Springer-Verlag, pp. 74-106.*, 1992.

[AMCK$^+$02] Jalal Al-Muhtadi, Roy Campbell, Apu Kapadia, Dennis Mickunas, and Seung Yi. Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments. In *Proceedings of The 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 74–83, 2002.

[AMCRC04] Jalal Al-Muhtadi, Shiva Chetan, Anand Ranganathan, and Roy Campbell. Super Spaces: A Middleware for Large-Scale Pervasive Computing Environments. In *Perware: IEEE International Workshop on Pervasive Computing and Communications, Orlando, Florida*, pages 198–202, March 2004.

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, July 1995.

[BDS01] Piero Bonatti, Ernesto Damiani, and Pierangela Samarati. A component-based architecture for secure data publication. In *Proceedings of 17th Annual Computer Security Applications Conference (ACSAC)*, pages 309–318, New Orleans, LA, December 2001.

[BE03] Christian Blum and Matthias Ehrgott. Local search algorithms for the k-cardinality tree problem. *Discrete Appl. Math.*, 128(2-3):511–540, 2003.

[Bis03] Matt Bishop. Computer Security: Art and Science. Addison-Wesley, ISBN 0-201-44099-7, 2003.

[BN89] David F. C. Brewer and Michael J. Nash. The Chinese Wall security policy. In *Proceedings IEEE Symposium on Security and Privacy*, pages 206–214, May 1989.

[Bra00] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*. MIT Press, 2000.

[BRKM91] Kenneth M. Butler, Don E. Ross, Rohit Kapur, and M.Ray Mercer. Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams. In *Proceedings of the 28th conference on ACM/IEEE Design Automation*, pages 417–420, San Francisco, CA, June 1991.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[BS02] Piero A. Bonatti and Pierangela Samarati. A uniform framework for regulating service access and information release on the Web. *Journal of Computer Security*, 10(3):241–271, 2002.

[BW96] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Computers*, 45(9):993–1001, September 1996.

[CE86] David Chaum and Jan-Hendrik Evertse. A secure privacy preserving protocol for transmitting personal information between organizations. In *CRYPTO*, 1986.

[CGP00] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[CKP00] U. Chajewska, D. Koller, and R. Parr. Making Rational Decisions using Adaptive Utility Elicitation. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pages 363–369, aug 2000.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient non-transferable anonymous multishow credential system with optional anonymity revocation. In *EUROCRYPT*, 2001.

[CL05] Jan Camenisch and Anna Lysyanskaya. A Formal Treatment of Onion Routing. In *Proceedings of CRYPTO 2005*, pages 169–187. Springer-Verlag, LNCS 3621, August 2005.

[CLM$^+$00] Roy H. Campbell, Zhaoyu Liu, M. Dennis Mickunas, Prasad Naldurg, and Seung Yi. Seraphim: Dynamic Interoperable Security Architecture for Active Networks. In *OPENARCH 2000*, Tel-Aviv, Israel, March 2000.

[CRB01] Ranveer Chandra, Venugopalan Ramasubramanian, and Kenneth P. Birman. Anonymous Gossip: Improving Multicast Reliability in Mobile Ad-Hoc Networks. In *Proceedings of the The 21st International Conference on Distributed Computing Systems*, page 275, Washington, DC, USA, 2001. IEEE Computer Society.

[CS00] David Coppit and Kevin J. Sullivan. Galileo: A tool built from mass-market applications. In *Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland*, pages 750–753, June 2000.

[DD02] Stéphane Demri and Deepak D'Souza. An Automata-Theoretic Approach to Constraint LTL. In *FST TCS '02: Proceedings of the 22nd Conference Kanpur on Foundations of Software Technology and Theoretical Computer Science*, pages 121–132, London, UK, 2002. Springer-Verlag.

[Epp94] David Eppstein. Finding the k shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. IEEE, November 1994.

[ET01] Uwe Egly and Hans Tompits. Proof-complexity results for nonmonotonic reasoning. *ACM Trans. Comput. Logic*, 2(3):340–387, 2001.

[FHJM94] Matteo Fischetti, Horst W. Hamacher, Kurt Jörnsten, and Francesco Maffioli. Weighted k-Cardinality Trees: Complexity and Polyhedral Structure. *Networks*, 24:11–21, September 1994.

[FK92a] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *In Proceedings of the 15th NIST-NSA National Computer Security Conference, Baltimore, MD, Oct*, 1992.

[FK92b] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *Proc. 15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, October 1992.

[FM02] Michael J. Freedman and Robert Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[FMK91] Masahiro Fujita, Yusuke Matsunaga, and Taeko Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *Proceedings of the conference on European Design Automation*, pages 50–54, Amsterdam, February 1991. IEEE Computer Society Press.

[FTA81] Fault Tree Handbook, NUREG-0492. United States Nuclear Regulatory Commission, 1981.

[Gai] Gaia, active spaces for ubiquitous computing. `http://gaia.cs.uiuc.edu/`.

[GGF98] Virgil D. Gligor, Serban I. Gavrila, and David F. Ferraiolo. On the formal definition of seperation-of-duty policies and their composition. In *In Proceedings of the IEEE Symposium on Research in Security and Privacy. (Oakland, CA.), 172–183*, 1998.

[Gra] Jonathan Graehl. `kbest`, a C++ library for efficiently finding the k shortest paths in a graph. Available from `http://jonathan.graehl.org/kbest.zip`.

[GSSS02] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing Magazine*, 1(2):22–31, April–June 2002.

[HMS03] John E. Hershberger, Matthew Maxel, and Subhash Suri. Finding the k shortest simple paths: a new algorithm and its implementation. In *Proceedings, 5th Workshop Algorithm Engineering & Experiments (ALENEX)*. SIAM, January 2003.

[Hon05] Jason I. Hong. *An Architecture for Privacy-Sensitive Ubiquitous Computing*. PhD thesis, University of California at Berkeley, Computer Science Division, 2005.

[hot] Hotmail Homepage. `http://www.hotmail.com/`.

[HY97] Takashi Horiyama and Shuzo Yajima. Exponential lower bounds on the size of OBDDs representing integer divistion. In *Proceedings ISAAC*, pages 163–172, 1997.

[J$^+$02] Bilel Jamoussi et al. Constraint-Based LSP Setup using LDP. RFC 3212, January 2002.

[Jaf84] Jeffrey M. Jaffe. Algorithms for Finding Paths with Multiple Constraints. *Networks*, 14:95–116, 1984.

[JFW02a] Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces project: Experiences with ubiquitous computing environments. *IEEE Pervasive Computing magazine*, 1(2):67–74, April–June 2002.

[JFW02b] Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing Magazine*, 1(2):67–74, April–June 2002.

[KHJ03] Håkan Kvarnström, Hans Hedbom, and Erland Jonsson. Protecting security policies in ubiquitous environments using one-way functions. In D.Hutter et al., editors, *Security in Pervasive Computing 2003*, volume 2802 of *LNCS*, pages 71–85. Springer-Verlag, Heidelberg, 2003.

[KMR93] David Karger, Rajeev Motwani, and G. D. S. Ramkumar. On Approximating the Longest Path in a Graph. In *Proceedings of WADS*, pages 421–432, 1993.

[KNC04] Apu Kapadia, Prasad Naldurg, and Roy H. Campbell. Routing with Confidence: Supporting Discretionary Routing Requirements in Policy Based Networks. In *Proceedings IEEE 5th International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, pages 45–54, June 2004.

[KNC05] Apu Kapadia, Prasad Naldurg, and Roy H. Campbell. Unlinkability through Access Control: Respecting User-Privacy in Distributed Systems. *Technical Report, University of Illinois, UIUCDCS-R-2005-2621*, August 2005.

[Kor02] Richard E. Korf. Search techniques. In Hossein Bidgoli, editor, *Encyclopedia of Information Systems*. Academic Press, San Diego, CA, August 2002.

[KSC04] Apu Kapadia, Geetanjali Sampemane, and Roy H. Campbell. KNOW why your access was denied: Regulating feedback for usable security. In *Proceedings of the ACM Conference on Computers and Communication Security (CCS)*, pages 52–61, Washington, DC, October 2004.

[LaV06] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, Available from `http://msl.cs.uiuc.edu/planning/`, 2006.

[LBT04] Ninghui Li, Ziad Bizri, and Mahesh V. Tripunitara. On Mutually-Exclusive Roles and Separation of Duty. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS), October*, 2004.

[LGP$^+$90] Douglas B. Lenat, Ramanathan V. Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. Cyc: toward programs with common sense. *Commun. ACM*, 33(8):30–49, 1990.

[LL01] Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.

[LN99] Jørn Lind-Nielsen. BuDDy – a binary decision diagram package. Technical Report IT-TR: 1999-028, Technical University of Denmark, 1999. Available from `http://www.itu.dk/research/buddy/index.html`.

[LRSW99] Anna Lysyanskaya, Ronald Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas of Cryptography, Volume 1758 LNCS*, 1999.

[LS98] Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[MdS04] Deborah L. McGuinness and Paulo Pinheiro da Silva. Explaining Answers from the Semantic Web: The Inference Web Approach. *Web Semantics: Science, Services and Agents on the World Wide Web Special issue: International Semantic Web Conference 2003*, 1(4):397–413, October 2004.

[Min04] Naftaly H. Minsky. A Decentralized Treatment of a Highly Distributed Chinese-Wall Policy. In *Proceedings IEEE 5th International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, pages 181–184, June 2004.

[MP92] Zohar Manna and Amir Pnueli. The temporal logic of reactive and concurrent systems. *Springer-Verlag*, 1992.

[MU04] Nenad Mladenović and Dragan Urošević. Variable neighborhood search for the k-cardinality tree. pages 481–500, 2004.

[OM87] Pekka Orponen and Heikki Mannila. On approximation preserving reductions: Complete problems and robust measures. Technical Report C-1987-28, University of Helsinki, Dept. of Computer Science, 1987.

[oxy] MIT Project Oxygen Homepage. `http://oxygen.lcs.mit.edu/`.

[P$^+$04] Andreas Pfitzmann et al. Anonymity, Unobservability, Pseudonymity, and Identity Management - A Proposal for Terminology. Draft v0.21, September 2004. Available from `http://dud.inf.tu-dresden.de/Literatur_V1.shtml`.

[PM04] Andreas Pashalidis and Chris J. Mitchell. Limits to anonymity when using credentials. In *Proceedings of the 12th International Workshop on Security Protocols, Springer-Verlag LNCS, Berlin*, Cambridge, UK, April 2004.

[PS95] Shipra Panda and Fabio Somenzi. Who are the variables in your neighborhood. In *Proc. International Conference on Computer-Aided Design (ICCAD '95)*, pages 74–77, San Jose, CA, November 1995.

[PV03] Pino Persiano and Ivan Visconti. An Anonymous Credential System and a Privacy-Aware PKI. In *R. Safavi-Naini and J. Seberry, editors, Information Security and Privacy, 8th Australasian Conference, ACISP 2003, volume 2727 of Lecture Notes in Computer Science. Springer Verlag*, 2003.

[QYZS03] Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On Selfish Routing in Internet-Like Environments. In *Proc. of ACM SIGCOMM*, August 2003.

[RHC$^+$02] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. GaiaOS: A middleware infrastructure to enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, October–December 2002.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, June 1998.

[RSG98] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Copyright and Privacy Protection*, 16(4):482–494, 1998.

[San98] Ravi Sandhu. Transaction control expressions for separation of duties. In *Proceedings of the 4th Aerospace Computer Security Applications Conference*, 1998.

[sap] SAPHIRE Project Homepage. `http://saphire.inel.gov/`.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[SCRD04] Atul Singh, Miguel Castro, Antony Rowstron, and Peter Druschel. Defending against Eclipse attacks on overlay networks. In *Proceedings of the 11th ACM SIGOPS European Workshop, Leuven, Belgium*, September 2004.

[SJW02] Oleg Sheyner, Somesh Jha, and Jeannette M. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA*, May 2002.

[SL02] Morris Sloman and Emil Lupu. Security and Management Policy Specification. *Special Issue on Policy-Based Networking*, 16(2), March 2002.

[SNC02] Geetanjali Sampemane, Prasad Naldurg, and Roy H. Campbell. Access control for Active Spaces. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 343–352, Las Vegas, NV, December 2002.

[SS75] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, September 1975.

[ST86] Robin A. Sahner and Kishor S. Trivedi. A Hierarchial, Combinatorial-Markov Model of Solving Complex Reliability Models. In *Proceedings of 1986 ACM Fall Joint Computer Conference*, pages 817–825, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[SV02] Stefano Salsano and Luca Veltri. QoS Control by Means of COPS to Support SIP-Based Applications. *Special Issue on Policy-Based Networking*, 16(2), March 2002.

[SZ97] Richard T. Simon and Mary Ellen Zurko. Separation of duty in role-based environments. In *IEEE Computer Security Foundations Workshop*, pages 183–194, 1997.

[Tar72] Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[UJ03] Jeffrey L. Undercoffer and Anupam Joshi. *Data Mining, Semantics and Intrusion Detection: What to dig for and Where to find it*. MIT Press, December 2003.

[ver] Verisign Homepage. `http://www.verisign.com/`.

[Ver01] Eric R. Verheul. Self-Blindable Credential Certificates from the Weil Pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 533–551. Springer-Verlag, 2001.

[WC96] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 14:1228–1234, 1996.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, September 1991.

[WL04] William H. Winsborough and Ninghui Li. Safety in automated trust negotiation. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 147–160, Oakland, CA, May 2004. IEEE Press.

[WSS$^+$01] Andrea Westerinen, John Schnizlein, John Strassner, Mark Scherling, Bob Quinn, Shai Herzog, An-Ni Huynh, Mark Carlson, Jay Perry, and Steve Waldbusser. Terminogy for Policy-Based Management. RFC 3198, November 2001.

[Yee02] Ka-Ping Yee. User Interaction Design for Secure Systems. In *Proceedings of the 4th International Conference on Information and Communications Security*, pages 278–290. Springer-Verlag, 2002.

[Yen71] Jin Y. Yen. Finding the K shortest loopless paths in a network. In *Management Science*, volume 17, pages 712–716, 1971.

[Yen72] Jin Y. Yen. Another algorithm for finding the K shortest loopless network paths. In *Proceedings of 41st Mtg. Operations Research Society of America*, volume 20, 1972.

[YNK01] Seung Yi, Prasad Naldurg, and Robin Kravets. Security-Aware Ad Hoc Routing for Wireless Networks. Poster presentation, ACM Symposium on Mobile Ad Hoc Networking & Computing (Mobihoc), 2001.

[YNK02] Seung Yi, Prasad Naldurg, and Robin Kravets. Integrating Quality of Protection into Ad Hoc Routing Protocols. In *The 6th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI), Orlando, Florida*, August 2002.

[YW03] Ting Yu and Marianne Winslett. A unified scheme for resource protection in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and privacy*, pages 110–122, May 2003.

[YWS03] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies in Automated Trust Negotiation. *ACM Transaction on Information and System Security*, February 2003.

[ZDE$^+$93] Lixia Zhang, Stephen E. Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 7(5):8–18, September 1993.

[ZK03] Terry Zimmerman and Subbarao Kambhampati. Learning-assisted automated planning. *AI Magazine*, 24(2):73–96, 2003.

[ZS96] Mary Ellen Zurko and Richard T. Simon. User-centered security. In *Proceedings of the Workshop on New Security Paradigms (NSPW)*, pages 27–33, Lake Arrowhead, CA, September 1996.

[ZSvR02] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A Secure Distributed On-line Certification Authority. *ACM Transactions on Computer Systems*, 20(4):329–368, November 2002.

Author's Biography

Apu Kapadia was born in [redacted] He grew up in Bombay, India and began his undergraduate studies in Computer Engineering at the University of Bombay. He transferred to the Department of Computer Science, University of Illinois at Urbana-Champaign in January 1996, where he subsequently obtained his B.S., M.S, and Ph.D. degrees in May 1998, May 2001, and October 2005 respectively.

Apu received a four-year High-Performance Computer Science Fellowship from the Department of Energy in August 2001 for his dissertation research related to trustworthy communication. During his graduate studies, Apu interned at the Los Alamos National Laboratory for two summers performing research on network protocols. After receiving his doctorate, Apu joined Dartmouth College as a Post-Doctoral Research Fellow with the Institute of Security Technology Studies.

Apu's research interests include security and privacy for heterogeneous systems, and the use of formal methods to reason about related problems.