POLICY-CONTROLLED EMAIL SERVICES

by

Saket Kaushik
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of the
Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

[illegible] Dr. Paul Ammann, Dissertation Co-Director

[illegible] Dr. Duminda Wijesekera, Dissertation Co-Director

Sanjeev Setia Dr. Sanjeev Setia Committee Member

[illegible] Dr. Hassan Gomaa Committee Member

[illegible] Dr. Daniel Menasce, Associate Dean for Research and Graduate Studies

[illegible] Dr. Lloyd Griffiths, Dean, The Volgenau School of Information Technology and Engineering

Date: 12/6/07 Fall Semester 2007
George Mason University
Fairfax, VA

Policy-Controlled Email Services

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Saket Kaushik
Master of Science
George Mason University, 2001
Bachelor of Technology
Indian Institute of Technology, Bombay, 1999

Directors: Dr. Paul Ammann, Professor
Dr. Duminda Wijesekera, Professor
Department of Information Technology

Fall Semester 2007
George Mason University
Fairfax, VA

Dedication

To my parents, for all of their support and guidance throughout my life. And to Shwetlena Sabarwal, my soul-mate, my inspiration, my friend.

Acknowledgments

It is a pleasure to thank the many people who made this thesis possible.

It is difficult to overstate my gratitude to my Ph.D. supervisors, Dr. Duminda Wijesekera and Dr. Paul Ammann. With their enthusiasm, their inspiration, and great efforts to explain things clearly and simply, they helped to make mathematics fun for me. It has been a great pleasure to work closely with Dr. Wijesekera. Throughout my thesis-writing period, he provided encouragement, sound advice, good teaching, good company, and lots of good ideas. I would have been lost without him. I would also like to express my sincerest gratitude towards Dr. William Winsborough for his efforts to improve my technical writing skills.

I am grateful to the secretaries in the ISE department, for helping the department to run smoothly and for assisting me in many different ways. Dolores Izer-Horn and Lisa Nolder deserve special mention.

I wish to thank my wife Shwetlena Sabarwal for providing a loving environment for me and unwavering support, in spite of my idiosyncrasies, postponements, *etc.*

Lastly, and most importantly, I wish to thank my parents, Manju Kaushik and Braham Datt Kaushik. They bore me, raised me, supported me, taught me, and loved me. My sister Vartika Mishra and brother-in-law Rajat Mishra's support proved very valuable in completing this thesis. To them and my wife I dedicate this thesis.

Table of Contents

Page

Abstract xi

1 Introduction 1

1.1 Motivation 1

1.1.1 A simple description of message delivery protocol 3

1.1.2 The message acceptance decision 4

1.1.3 The delivery prioritization problem 9

1.1.4 Constraint logic programming approach 10

1.1.5 Pragmatic concerns 10

1.2 The Thesis statement 11

1.3 Summary of claims 11

1.4 Published work 12

2 Current Email Transmission Standards 15

2.1 Introduction to Email delivery 15

2.2 Messages and their structure 15

2.2.1 Message Envelope 16

2.3 Simple Mail Transfer Protocol (SMTP) 18

2.3.1 SMTP Transmission 20

2.3.2 Mail Transfer Agents (MTAs) 21

2.3.3 Misuse Cases 28

2.4 Chapter conclusion 30

3 Email Control Policy Architecture 31

3.1 Introduction 31

3.2 Policy architecture 34

3.2.1 Sender's Send Policy(SP) 34

3.2.2 SESPs Postman Policy (SPP) 35

3.2.3 Service Level Agreement Policy (SLAP) 36

3.2.4 Message Scheduling Policy (MSP) 37
3.2.5 RESPs Postman Policy (RPP) 39
3.2.6 Mailbox Resource Allocation Policy (MRAP) 40
3.2.7 Message Negotiation 40
3.2.8 Privacy 42
3.3 Policy Composition and Implementation 43
3.4 Chapter conclusion 45
4 The Formal Model of Policies 47
4.1 Introduction 47
4.2 The Formal Model 52
4.2.1 Syntax 53
4.2.2 Encoding emails into the constraint domain 57
4.2.3 Semantics 58
4.3 Message refinement 61
4.3.1 Feasible refinements 63
4.3.2 Selection of a suitable fix 63
4.3.3 Policy control over feedback 64
4.4 Examples 65
4.5 Evaluation and Complexity 68
4.6 Chapter conclusion 71
5 A Formal Model for Generalized Messages 72
5.1 Introduction 72
5.2 Formal Model 74
5.2.1 Syntax 74
5.2.2 Semantics 83
5.3 Materialization Structure 88
5.4 Chapter conclusion 89
6 Privacy 91
6.1 Introduction 91
6.1.1 Overview of our approach 94
6.2 Examples 95
6.2.1 Direct disclosure 95
6.2.2 Disclosure through leaky mechanisms 96
6.3 Formal Model 98

6.3.1 Syntax 98
6.3.2 Semantics 100
6.4 The Attacker Model 101
6.4.1 Capabilities 102
6.4.2 Scripting an attack 104
6.5 Policy transformations for privacy 106
6.5.1 Transformation algorithm 106
6.5.2 Syntactic Properties 111
6.5.3 Semantic Properties 112
6.5.4 Protection achieved from transformed policies 123
6.6 Chapter conclusion 125
7 Extensions to SMTP 128
7.1 Introduction 128
7.2 Extensions to SMTP 128
7.2.1 Requirements 129
7.2.2 Extensions 131
7.2.3 Extended SMTP state machine 132
8 Securing Email by Design – A Web Services-based Case Study 138
8.1 Introduction 138
8.2 Use cases and Misuse cases 139
8.3 Web Services for message transmission 142
8.3.1 Message types 143
8.3.2 Messages 145
8.3.3 WSEmail family of Web Services 148
8.4 BPEL orchestration of WebMail 152
8.4.1 SESP process specification 152
8.4.2 RESP process specification 155
9 Related Work 170
9.1 Introduction 170
9.1.1 Proposals 171
9.1.2 Current practices 182
10 Conclusion 185
Bibliography 188

List of Tables

Table Page

3.1 Control achieved through policies 34

4.1 Predicates 54

5.1 Required Local Predicates (RL) 75

5.2 Import-Export Predicates (IE) 76

5.3 Optional Local Predicates (OL) (Examples) 76

5.4 $\Phi^{V_i}_{P_{\text{MSP}}}$ calculation for message acceptance 86

5.5 $\Phi^{V_i}_{P_{\text{MSP}}}$ calculation for message revision 88

6.1 Leakage through SMTP reply codes 96

7.1 Proposed additions to SMTP reply codes 131

8.1 Basic types of message elements 143

8.2 Complex types of message elements 145

8.3 Types of messages 148

8.4 Acceptance Policy 163

List of Figures

Figure Page

2.1 SMTP ladder diagram 21
3.1 Principals and Policies in a Policy-mediated Email Pipe 32
3.2 SESP's Postman Policy (SPP) 35
3.3 Service Level Agreement Policy (SLAP) 36
3.4 Message Scheduling Policy (MSP) 37
3.5 Mailbox Resource Allocation Policy (MRAP) 39
3.6 Policy-Mediated Email Pipe 44
5.1 Interaction of policies for message transmission 73
6.1 Transformation algorithm 127
7.1 Service Level Agreement Policy (SLAP) 130
7.2 Modified extended handshake 133
7.3 Transmission with no feedback. 134
7.4 Transmission with MSP communicated on a per session basis. 135
7.5 Transmission with MSP shared per session as well as on a per message basis. 136
7.6 Transmission with MSP shared per session and per message and MRAP shared on a per message basis. 137
7.7 Transmission where only MRAP is shared on a per message basis. . . 137
8.1 Basic Types in WSDL 144
8.2 WSDL mail message 146
8.3 WSDL Application Data 147
8.4 WSDL Application Data (contd.) 149
8.5 WSDL Control Data 150
8.6 WSDL portType specs for SESP services 153
8.7 WSDL portType specs for SESP services(contd.) 154
8.8 WSDL portType specs for SESP services(contd.) 155

8.9 WSDL portType specs for RESP services 156
8.10 WSDL portType specs for RESP services(contd) 157
8.11 WSDL portType specs for RESP services(contd) 158
8.12 WSDL portType specs for third party services 159
8.13 Example SESP Process (part I) 159
8.14 Example SESP Process (contd.) (part II) 160
8.15 Example SESP Process (contd.) (part III) 161
8.16 Example SESP Process (contd.) (part IV) 162
8.17 Example RESP Process (part I) 164
8.18 Example RESP Process (contd.) (part II) 165
8.19 Example RESP Process (contd.) (part III) 166
8.20 Example RESP Process (contd.) (part IV) 167
8.21 Example RESP Process (contd.) (part V) 168

Abstract

POLICY-CONTROLLED EMAIL SERVICES
Saket Kaushik, PhD
George Mason University, 2007
Dissertation Directors: Dr. Paul Ammann and Dr. Duminda Wijesekera

The context for this research proposal is an area of work that seeks to replace the current state of access-control for email, in which an arbitrary message sender enjoys unregulated "append" access to message recipient's email mailbox, with a policy framework, in which each principal involved in a message exchange - namely the sender, the sender's service provider, the recipient's service provider, and the recipient, can articulate its interests for regulating access to resources under its control. Though there exist a vast number of automatic control techniques to limit transmission of email messages, specifically, to stop unwanted messages reaching a recipient, they are still prone to dropping some desirable messages. This often prompts recipients and other principals to relax the message acceptance requirements. This in turn makes them easy targets for sending commercial or fraudulent mail. We propose a novel scheme to overcome this handicap. Our scheme makes the transmission mechanism aware of the documentation required with a message to make it acceptable downstream. For instance, if a recipient wishes to receive only those messages that have a monetary guarantee, also known as a bond, then the transmission system must be made aware of this fact so that desirable messages can satisfy this requirement.

Consequently, recipients and other principals can express and enforce precise acceptance requirements, through explicit policies, and gain control over their resources.

In addition to the problem of enforcing precise acceptance requirements in the transmission process, there exists no means of flexibly combining available email-control techniques tailored to the needs of a particular recipient or its service provider. This is the primary reason for the inability to express requirements suited to a particular individual. For instance, currently it is not possible to state a requirement like 'allow messages, initiated by a human sender affiliated with *George Mason University*, even though the spam filter ranks them as possible spam'. In our view a policy-based approach is well-suited to attain this objective. The use of these control-techniques leads to significant deviation of behavior from what is prescribed in the current email delivery protocol. In other words, the protocol lacks significantly in representing the current delivery requirements. Clearly, it requires an overhaul to correspond to current requirements and reduce ambiguities during protocol play; a goal that we propose to research in this study.

We propose using constraint logic programming (CLP) to articulate and evaluate different types of policies. This is because the way messages are constructed and acceptance conditions are evaluated, a CLP approach seems a natural way to model these operations. In addition, CLP approach promises to simplify the task of providing feedback for rejected messages, so that they can be revised and retransmitted. Since declarative policies can describe control on a very high-level, we also propose to study refinements of these high-level directives to protocol level actions.

Chapter 1: Introduction

1.1 Motivation

The current email system is designed to transfer messages from one host to one or more hosts over the Internet in a reliable and robust manner. The unit of transmission is a mail message that is initiated by the message sender. Messages are transmitted asynchronously over a network through multiple hops till they reach their destination – the recipient's mailbox. The delivery infrastructure provides best effort delivery for all messages [1]. For example, the system ensures adequate provisions to prevent loss of messages, like multiple delivery attempts, *etc.*; and in the case of failure, the sender is appropriately notified through error messages. For a message sender such provisions are highly desirable. However, this is not an ideal scenario from the recipient's perspective. This is because any message initiated at anyone's behest would appear in the addressed recipient's mailbox. This renders the system highly susceptible to abuse by senders, making recipients easy targets of unwanted commercial messages.

Techniques for automatic identification (and hence removal) of unwanted messages abound. These solutions try to remove 'junk' from the recipient's mailbox, without requiring any human intervention. In a majority of cases, the solutions require some accompanying documentation, like monetary bond value or cryptographic signatures or even proof of human initiation, to distinguish between wanted and unwanted messages. We provide a comprehensive review of prominent techniques in § 2. However, most automatic classification techniques are fallible to dropping desirable messages. Part of the reason is that senders are able to effectively mask their messages as 'legitimate' communication making it difficult to automate the task of distinguishing a legitimate message from an illegitimate one. Incorrect categorization of legitimate messages as unwanted and subsequent prevention of transmission is termed as a *false positive*. Consequently, neither the recipient nor the sender is informed of the loss of communications.

To prevent losing messages in the above described manner, recipients often tend make the acceptance criteria more permissive. For example, if a particular email filter regularly drops important mail, then a recipient would tend to lower its sensitivity rather than losing out on receiving important messages. Consequently, recipients fail to use the full power of available control techniques. As a result, the recipient is susceptible to receiving a large number of unwanted messages.

In this proposal we plan to investigate ways by which the above handicap can be overcome. We propose to study a simple scheme of communicating feedback regarding rejected messages upstream. With feedback the delivery system can be made aware of why a message was rejected. In other words, the transmission mechanism is made aware of ways to change the accompanying documentation to get a message accepted downstream. For example, if a recipient wishes to receive only bonded messages, the sender must be made aware of this fact to be able to send any messages to the particular recipient. We assume that verifying the validity of accompanying documentation can be accomplished by the mechanism that consumes it. For example, the reliability of bond accompanying a message must be possible through financial escrow services. Of course, providing feedback for every rejected message runs the risk of leaking private information upstream and in turn help malicious entities. Therefore, in this research we develop ways through which feedback can be sanitized and still be helpful in getting good messages through to the recipient.

Next we introduce the message transmission process in § 1.1.1. We follow it with a characterization of the types of proposed control decisions and bring out their research issues in 1.1.2 and 1.1.3. We also motivate the related sub-problems, like privacy, refinement, *etc.*, in 1.1.2, that require investigation as well. Next we define the scope of this research, with a focus on the specific problems that we addressed.

1.1.1 A simple description of message delivery protocol

The current Internet mail system, or e-mail system, has three major components: *user-agents*, *mail servers* that communicate using the *Simple Mail Transfer Protocol (SMTP)* [1]. A typical message starts its journey from the sender's user agent, travels to the sender's mail server, and then travels to the recipient's mail server either directly or through intermediate mail servers. At the recipient's mail server it is deposited in the recipient's mailbox [2]. SMTP, defined in RFC 2821 [1] transfers messages from the senders mail server to the recipients mail server. First, the SMTP client (running on the sending mail server host) has TCP establish a connection on port 25 to the SMTP server (running on the receiving mail server host). If the server is down, the client tries again later. Once this connection is established, the server and client perform some application-layer handshaking. During this handshaking phase, the SMTP client indicates the email address of the sender and the email address of the recipient. Next, the client sends the message. The client then repeats this process over the same TCP connection if it has other messages to send to the server, otherwise it instructs TCP to close the connection [2].

The current protocol definition constrains the receiving mail server, also known as the *server*, to a *slave* of the sending mail server, also called the *client*. The client issues *commands* (*i.e.*, message related operations), which the server is obligated to respect. Only in exceptional cases or failure can the server refuse to execute the issued command. In such a case, the server is usually required to respond with a reason for its inability to respect client's command. Clearly, it's this master-slave relationship that needs to be revised for the server to gain control over its own resources. Existing email-control techniques achieve this aim only to a limited extent. In our proposal, we make a case for a more general and customizable control provided to servers. We discuss this issue in more detail next.

1.1.2 The message acceptance decision

The decisions regarding accepting messages, relaying them or storing them in the file-system can be viewed as access control decisions, where the decision maker allows 'append access' to a particular resource, be it a file system or a message queue. We term this problem of deciding on the acceptance of a message as *the message acceptance problem*. In the current system, the decision is almost always a 'yes', *i.e.*, accept, relay or file a message, whatever the case may be, except for the case where email-control techniques reject a message from being appended to the recipient mailbox. These control techniques are forms of *attribute based access control* mechanisms designed to protect the recipient mailbox. However, those messages that do not or cannot make into a mailbox waste network bandwidth and resources upstream. They should ideally be dropped earlier in the transmission process or revised to satisfy downstream email-control mechanisms. In this work we propose to generalize the attribute based access control scheme and extend the protection to most network resources through our policy-based control framework.

The primary task involved in solving the message acceptance problem is to decide whether to transmit or accept a message at each hop. Because a number of techniques are available to accomplish this task, we propose to leverage on them in solving the message acceptance problem. However, each individual proposal approaches this problem from a very narrow perspective. For example, a message bond technique [3] only considers the monetary signalling and economic consequences whereas human initiation tests [4] block machine-constructed messages. A recipient may choose to use any one of the proposals or mix and match them to serve his or her needs. For example, a recipient may choose digital signatures [5] for authentication with a feature or pattern matching based approach [6] to gauge the information content and so on. Currently there exists no means for combining, controlling and using various techniques in tandem. The challenge here lies in *bringing a majority of control-techniques under a single umbrella.* Although, some products combine a small subset of available techniques [7], it is done in a rigid, non-extensible manner. By putting the combination of various techniques under policy control, the proposed work will correct this problem and enable all techniques to be combined in arbitrary ways that are deemed appropriate at each site in the email network. In addition, generating the composition of the policies as an explicit construct allows the sender to answer questions such as "What attributes does a message need to receive a particular level of service", and the recipient to answer to questions such as "Given my privacy policy, what sort of mail will I be able to receive from particular senders or classes of senders."

Policy feedback, Policy communication and Privacy

Currently, a message is sent out blindly in the hope that it is satisfactory to the recipient and recipients have to sort through mail that, by and large, lacks useful tags for differentiation. Better decisions regarding construction and transmission of mail messages are enabled if pertinent information is pushed as near the sender of a message as possible, modulo the privacy considerations of downstream principals. These principals can then choose for authentication, reputation, bonding, or other services consistent with the recipient's requirements. Once the requirements for email delivery are expressed in a policy language, that policy, or parts of it, can be communicated. It can be published or otherwise made available upstream, enabling undesirable messages to be dropped at an earlier stage in the transmission route. For example, a policy may specify that a message from a strongly authenticated business partner will be accepted even if the message is ranked as likely spam by a Bayesian filter. If this is known by his email service provider (ESP), a partner can be given the opportunity to strongly authenticate. In this respect, the proposed approach differs markedly from most existing techniques by letting up-stream agents know what they can do to improve the likelihood that their messages reach their destination, whether it be using strong authentication, attaching a monetary bond, or improving their reputation. The proposed study plans to develop *policy feedback* and *policy communications* to share relevant information upstream.

Of course, not all such information is suitable for dissemination. For example, knowing filtering rules could assist in transmitting fraudulent messages. Part of the proposed research will develop techniques for automatically sanitizing policies in a semantically sound and precise manner so they can be securely shared. Furthermore, simply by enforcing a given policy, information about that policy can be made observable. Research reported in this dissertation develops techniques for closing high-bandwidth channels through which highly sensitive policy content could otherwise become known. We look at one particular combination of using blacklists/whitelists with message bonds. In this case, if the recipient requires messages to be bonded with a certain monetary value whenever the message sender is on their blacklist, the combination is subject to privacy leaks. This is because seizure of a bond can provide feedback to the sender that he or she is on the recipient's blacklist. We assume here that sender has a means to know how a message is accepted by a particular recipient. We can easily argue that such a situation is possible because a sender can send different types of messages, something akin to a dictionary attack, and find out exactly how the recipient email system behaves. Similarly, if a recipient uses filtering methods with message bonds or embedded custom URIs, again private information about the filters, such as efficacy of the filter, can be leaked through bond seizure or evidence of a visit to the web page.

Even though there are out of band channels for information leakage (like the recipient telephoning the sender), when sensitive information is leaked through the system itself, we term it as *unsafe.* When there is no possibility of leaking of sensitive information through the system, it is termed as *safe.* The problem of designing a system that is safe is termed as the *privacy problem.*

Policy refinement

Policy-based control permits separation of management and functionality concerns of the email network. In principal, the behavior of a user agent or a mail server can be adapted without re-implementing its functionality. However, policy specifications and their evaluations are expected to be expressed in a language that is at a higher level of abstraction than the operational language of the email system. For example, a policy evaluation can require that a particular message be 'rejected'. However, the SMTP implementations understand only SMTP commands and reply codes [1]. Enforcement of the 'reject' decision requires translation to some protocol-level *action* or a system-level *call*. In addition, the translated command has to be ascribed to a particular principal, like a mail server, to be effected. Policy refinement is this translation process that transforms higher level decisions to action items in the operational language of the system. In summary, there are three principal objectives in a policy refinement process [8]:

1. Determine the resources that are needed to satisfy requirements of the policy.
2. Translate high-level policies into operational policies that the system can enforce.
3. Verify that the lower level policies actually meet the requirements specified by the high-level policy.

Policy refinement specifications require a formal representation for objects, their behavior and their organization. Along with the formalisms, a refinement technique for resolving high-level goals into more concrete ones [8] is essential. To this end, we propose to develop a formalism to model the behavior and organization of relevant objects or actors and research on techniques for decomposing abstract requirements into more concrete ones.

1.1.3 The delivery prioritization problem

Solving the message acceptance problem to mitigate the problem of unwanted, annoying or fraudulent mail is essential. However, in spite of such a solution in place, scenarios can emerge in which the solution is not sufficient to ensure that desired messages reach the addressed recipients in time or reach them at all. As an example, consider a scenario where a computer virus outbreak floods an email network with a large number of infected messages. Because the current email system must either deliver or reject *all* the messages in the order in which they are queued, messages queued behind the large number of infected messages will get inordinately delayed. This is because each message has to be processed *in turn* to decide if it should be accepted, relayed or rejected. An email-bomb attack [9] is another scenario where a large number of messages addressed to a recipient can cause delay or loss of services to a particular recipient. Business requirements, like prioritizing CEO's or stock broker's or paid customers messages, and *fairness* issues also require prioritization schemes to be in place. Thus, the problem of prioritization of messages is separate from the problem of message acceptance. We term this problem as the *delivery prioritization problem.* In this proposal we plan to support message prioritization, however we don't subscribe or recommend any particular prioritization scheme.

1.1.4 Constraint logic programming approach

We propose to use a restricted form of constraint logic programming (CLP) language to specify our policies. This approach has many benefits. It provides a concise, logical specification of the messages that will receive mail service. It allows one to consider messages that would hypothetically be accepted, without fully specifying the message.

This will be very helpful in supporting negotiation of message fixes. It will also enable one to examine the classes of messages that are accepted, either by a single policy or by a composition of various policies in our framework. Furthermore, by using a logic programming-based approach, we enable ourselves to adopt well-known techniques for resolving conflicts between accept and reject directives in the policy, and to ensure that the policy decides the disposition of all messages [10].

1.1.5 Pragmatic concerns

Apart from the basic technical requirements, practical issues also govern many design issues. Due to a large number of installations of the current SMTP, a completely new protocol replacing the current SMTP protocol is clearly infeasible for adoption. In fact, the only hope for a new system to be adopted is in designing an incrementally adoptable and backward compatible system that requires minimal changes to SMTP. SMTP features, such as reliability and robustness, can be useful and could be accommodated in the revision. It is unreasonable to expect a typical user to directly write CLP policies. To aid the reader/users in formulating CLP policies, we present example policies. In addition, related research by Fages [11] gives us hope that CLP policies could be used by typical users. In this dissertation we pursue mechanisms that can be designed such that adoption concerns are addressed.

1.2 The Thesis statement

> “Policy controlled email services with sanitized feedback make it possible to securely reduce false positives in email control mechanisms while accommodating current and future mechanisms for providing control and

feedback and avoiding the need to replace current email infrastructure."

1.3 Summary of claims

This dissertation proposal makes several claims. In this section we list them to provide the base set of claims that we need to verify in the experimenting and testing phase.

Claim 1. *With the proposed policy framework, most of the existing and future email-control techniques can be supported, and used in combination.*

Claim 2. *With additional controls to reject or delay messages, downstream principals in an email pipe will be able to reduce the bandwidth and number of unwanted messages delivered to recipients.*

Claim 3. *If feedback is provided for rejected, but desirable, messages, it will lead to less chances of loss of desirable communications.*

Claim 4. *If downstream acceptance preferences are made known upstream, it will result in transmission of messages that have the required documentation for satisfying downstream preferences.*

Claim 5. *Feedback for rejected messages and communicated policies can be sanitized before transmission, such that they do not leak any sensitive information.*

Claim 6. *Policy evaluation, policy communication and policy feedback schemes can be integrated into an extension of SMTP protocol.*

1.4 Published work

Portions of the proposed study have been published in refereed conferences and workshops, and a mapping of claims to the publications is presented next.

Policy Framework: Saket Kaushik, Paul Ammann, Duminda Wijesekera, William Winsborough and Ronald Ritchey. A Policy Driven Approach to Email Services. In *Proceedings of IEEE 5th International Workshop on Policies for Distributed Systems and Networks (Policy 2004)*, p: 169–178, New York, June 2004

- This paper presents a preliminary policy framework design for solving the message acceptance decision problem. It includes discussions on the MSP, SLAP and MRAP policies and simple extensions to SMTP protocol to accommodate them.

Policies as module networks: Saket Kaushik, Duminda Wijesekera, William Winsborough and Paul Ammann. Distributed CLP clusters as a security policy framework for email. In *1st international workshop on Applications of constraint satisfaction and programming to computer security (CPSec)*, (at CP 2005), pages 31–45, Barcelona, Spain.

- This paper presents a communication scheme where messages are converted to logical predicates that are *imported or exported* between principals. Message acceptance decision is implemented by a policy module which evaluates the acceptance criteria against the imported predicates and in-turn exports relevant predicates – downstream in the case of an accept decision, and upstream in the case of a reject decision. This paper also extends the policy framework to include policies for solving the delivery prioritization problem.

Policy extensions to SMTP with PTIME evaluation: Saket Kaushik, William Winsborough, Duminda Wijesekera and Paul Ammann. Email Feedback: A

Policy-based Approach to Overcoming False Positives. In *3rd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code (FMSE 2005)*, Fairfax, VA, U.S.A.

- This paper generalizes communication scheme to include transmission of policies upstream to aid the process of 'customizing' messages for the intended recipient. We show that under specific assumptions, the evaluation of acceptance criteria can be completed in PTIME (polynomial in the size of the policy). In addition, we relate the minimum possible changes required in the SMTP state machine to accommodate the generalized communication scheme.

Prevention of leakage of sensitive information: Saket Kaushik, William Winsborough, Duminda Wijesekera and Paul Ammann. Policy Transformations for Preventing Leakage of Sensitive Information in Email Systems. In *20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 06),* Sophia Antipolis, France.

- This paper analyzes several ways in which recepients' sensitive information be leaked to senders by specially crafted email messages. We present an attacker model and develop a program transformation technique that can be used to prevent leakage of sensitive information.

Secure WebMail: Saket Kaushik, Duminda Wijesekera and Paul Ammann. BPEL Orchestration of Secure WebMail. In *2006 ACM Workshop on Secure Web Services (SWS 2006)*, Fairfax VA, November 2006.

- This paper presents an implementation profile for a secure Web Services based Email system. We present the implementation details for achieving majority of SMTP use cases, while preventing several misuse cases discussed in this thesis.

Chapter 2: Current Email Transmission Standards

2.1 Introduction to Email delivery

This chapter briefly reviews the process and the components of the current email transmission infrastructure. We begin with a discussion of the structure of a mail message. Next, we review the details of current SMTP protocol and sample SMTP conversations. Following this discussion, we delve deeper into the working of SMTP mail servers. Lastly, we illustrate typical SMTP Use Cases and Misuse Cases.

2.2 Messages and their structure

Much like its physical counterpart, an internet message or email consists of an envelope and its contents (or the body of the message), as defined in RFC 2822 [12, 13]. However, email message envelopes contain much more information than the physical mail envelopes. The main purpose of an envelope is to route a message from its source to its destination. In addition, it may record information about its contents, like the size of the *objects*, their relevance, urgency, type, *etc.* These additional *attributes* are of immense use downstream, as will be shown later. Content portion of a message comprises of *objects* to be delivered to the recipient. These objects are defined [14] to be of specific types and are popularly known as MIME objects. For the rest of this work we treat the content portion (with, possibly, multiple MIME objects) as a single entity as the number of objects and their structure is not critical to our study.

More details will be presented in chapter 4. However, the structure of the envelope is of utmost importance to our study and we review it next.

2.2.1 Message Envelope

A message, envelope and contents included, is a sequence of 'US-ASCII' (ASCII) characters. Contents that use characters outside the range of ASCII domain must be encoded to an ASCII format at the source and decoded back into their native format at the destination. An envelope consists of 'header fields' that are pairs of 'field names' names and 'field values'. Field names themselves are drawn from a standard, but extensible, set of names, while the domain of values is a finite domain of sequence of ASCII characters. All the header fields together are referred to as the message header or simply as the header. Header fields at times are loosely referred to as the named header, *e.g.*, 'from header' is a header field with 'FROM' as the field name. An envelope is a 'set' of headers, but at times it may be a 'multi-set' or a 'bag' of headers. That is, some header fields may be repeated in the envelope. A legal message must have the set of mandatory headers; it can also contain optional headers. Content, or the body of the message is optional as well.

Syntactic details concerning the format of message headers is not critical for later analysis, but, it is important to note that a special character sequence ('CRLF' – carriage return followed by a linefeed) is used to distinguish different parts of a message. In other words, CRLF delimiters separate one header from another and the body of the message. Number of header fields is not fixed, however certain header fields are mandatory and must be present in a legal message. These fields include the following:

- `from` – maximum one occurrence per message.

- `orig-date` – maximum one occurrence per message.

In addition, from the three fields: `to, cc` and `bcc` at least one should be present. If present, maximum one occurrence per field is allowed. Many other options fields can be present in a message, like, `subject, message-id,` *etc.* Informally, the `from` field identifies the message sender's email address, `orig-date` identifies the time of message origination, `to` identifies the recipient's email address, *etc.* Interested reader is referred to RFC 2822 [12] for the complete semantics of header fields. For our purposes, we assume the presence of following types of header fields (as explained in RFC 2822).

Originator fields A set of header fields that identify the source or originator of a message. This set usually contains the fields: `from, sender` or `reply-to` headers.

Destination fields A set of header fields that identify the recipient(s) of the message. This set usually contains the fields: `to, cc` or `bcc` headers.

Identification fields A set of headers that uniquely identify an email message correspondence between the sender and the recipient. This set usually contains the fields: `message-id, in-reply-to, references, msg-id,` *etc.*, headers.

Informational fields These fields are optional and provide additional information regarding the body of the message. Typically this set includes following headers: `subject, comments, keywords`.

Optional fields Optional fields, as the name suggests, are optional and are currently unspecified. They add flexibility of including information not covered in fields discussed above. Senders and recipients can chose to include fields understood at either ends for additional information transfer.

The above description does not include fields of the type 'trace fields' and 'resent fields' discussed in RFC 2822 because we don't explicitly leverage on them in this work. However, these fields are useful in the actual delivery of a message because of two reasons. First, it is possible to make routing decisions based on the origin and path taken by a message to reach a decision point. Secondly, in case of delivery failure reverse paths are essential for informing the senders about the same. However, we analyze the stated problems at a higher level than these low level details and assume that required functionality is implicitly available. On the other hand, optional fields play an important role for solving the problems earlier stated and we delve on them in greater detail later. Next we discuss the current standards used for transmission of messages over the Internet.

2.3 Simple Mail Transfer Protocol (SMTP)

Simple Mail Transfer Protocol, *a.k.a.*, SMTP [1] is the default protocol used for email transport. Its objective is to define standards for reliable and efficient delivery of messages between sending and receiving applications. Moreover, message transmission is accomplished asynchronously (*i.e.*, persistent asynchronous communication [15]), and usually requires multiple hops [1], *i.e.*, the message is routed through multiple intermediate servers before finally reaching its destination.

RFC 2821 describes four participants involved in a message transmission in three hop message transmission model as follows.

Sender/File System The sender process/file system where the message originates.

SMTP Client Message is routed to a process that can engage in SMTP *conversations* with another process. This process is handed the responsibility of locating the destination of the message and transmitting the message to the destination domain. If message delivery fails, the SMTP Client is required to report it to the Sender process/File System.

SMTP Server SMTP Server process receives messages delivered by SMTP Client process. Client and Server are alternatively referred to as *Mail Transfer Agents* (MTAs) and more generally as *mail servers*.

Recipient/File System The recipient process/file system where the message is finally delivered/stored.

Senders are free to deliver messages to the SMTP Client using a means other than SMTP transmission. Common techniques include Web-based delivery, *etc.* Similarly, other means, like Post Office Protocol [16] or Internet Message Access Protocol [17], may be used by the SMTP Server to deliver an incoming message to the recipient. An SMTP Client is expected to use the Domain Name Service (DNS) [18] to locate an SMTP Server. Therefore, message transmission may be expected to be completed in a single SMTP session between originating Client and destination Server. However, some destination Servers may only be accessible through public *gateways*. Therefore, RFC 2821 allows the possibility of more than three hops as described above. That is, the SMTP Server may not be the final destination for the message and the message may be handed over to another SMTP Server by the current SMTP Server (through another SMTP conversation in which the current SMTP Server behaves as the SMTP

Client). In addition to being transferred the responsibility of delivering the message, the SMTP server is required to report delivery failure to the SMTP Client.

2.3.1 SMTP Transmission

An SMTP transmission between a SMTP Client and a SMTP Server is completed in three phases: the *handshake* phase, the *mail transaction* phase and the *termination* phase. SMTP transmission is initiated and dictated by the SMTP Client, with the SMTP Server following with a standard reply to each standard SMTP command issued by the Client. The Server is obliged to *accept* (*i.e.*, obey) each command issued by the Client whenever it can. That is, a Server can refuse a Client's command only in the cases of transient or fatal errors. SMTP sessions are torn down by the Client, usually when it runs out of commands to issue. A simplified version of SMTP transmission is shown in figure 2.1. The figure shows a session being initiated by the Client and followed by the Server. There are four distinct parts shown here (instead of three). The first part is where the Client initiates the handshake. After the handshake the Client begins the mail transmission phase, composed of two subparts – *header transmission* and *body transmission*. During a single SMTP session multiple message transmissions are possible, *i.e.*, header and body transmission phases can be repeated multiple times. Lastly, after all messages have been processed (*i.e.*, transmitted or have transient or fatal errors), the Client initiates session termination phase. Because multiple messages are processed in an SMTP session, they are required to be stateful with both the mail servers maintaining a common view of the current state. For complete syntax and semantics of the protocol, the reader is referred to RFC 2821 [1].

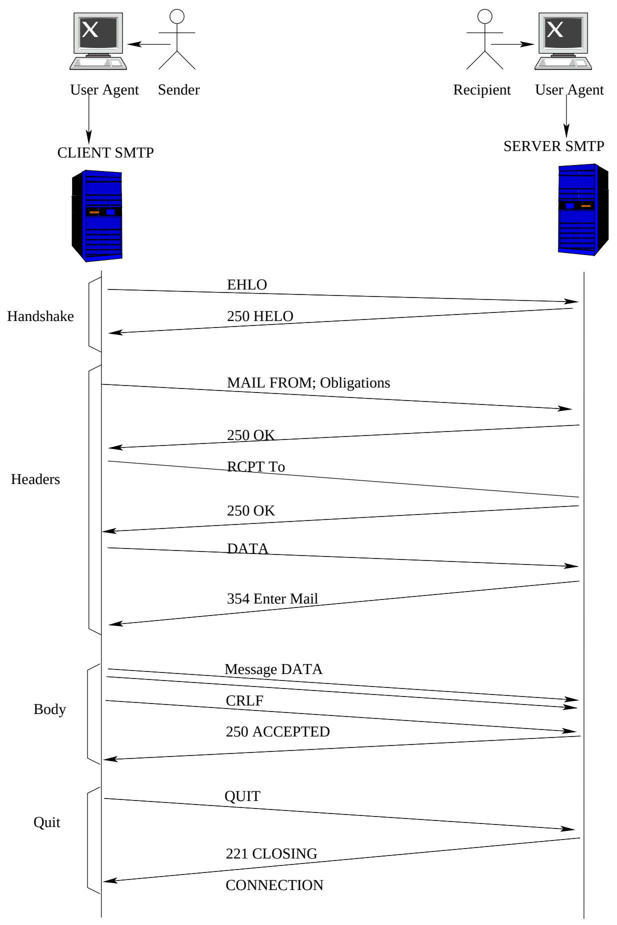

Figure 2.1: SMTP ladder diagram

2.3.2 Mail Transfer Agents (MTAs)

Next we briefly illustrate the main functions of SMTP Client and SMTP servers that take part in SMTP transmissions as described in figure 2.1. Pseudo code process 1 outlines SMTP Server's state changes during message transmissions.

PROCESS 1: SMTP Server

```
1: state = not connected;
2: while true do
3:     Wait for request from client; // initiation
```

```
4:  if request==‘EHLO’ AND state == not connected then
5:     state =connected; // change state to connected
6:     Send (250 OK) response; // acknowledge the state change
7:     Wait for request;
8:  else
9:     state = not connected; // for all other commands maintain state
10: end if
11: if request==‘MAIL TO’ AND state == connected) then
12:    state = received sender;
13:    Send (250 OK) response;
14:    Wait for request;
15:    if request ==‘RCPT TO’ AND state == received sender then
16:       state =received recipient;
17:       Send (250 OK) response;
18:       Wait for request;
19:    end if
20: end if
21: if request == ‘DATA’ AND state == received recipient then
22:    state = receiving data;
23:    Send (334 Enter Mail) response;
24:    Wait for request;
25: end if
26: if request ==‘CRLF’ AND state == receiving data then
27:    state = received data;
```

```
28:       Send (250 OK) response;
29:       Wait for request;
30:    end if
31:    if request ==‘QUIT’ AND state == received data then
32:       state = not connected;
33:       Send (221 Connection closing) response;
34:    end if
35: end while
```

Process 1 above presents a simplified algorithm for the state changes maintained by the SMTP Server. A similar algorithm for SMTP Client is presented below in Process 2. Lines 4 – 8 (process 1) and lines 9 – 12 (process 2) show the *handshake* phase between the Server and the Client. Lines 11 onwards (process 1) shows the state changes at Server after an SMTP session (connection) has been set up by a Client. These lines also show how a subservient Server responds to the Clients commands. Note that the shown processes assume that the communicating party is *error-free* and follows the protocol. This is the reason it does not check for *out of order messages* or *message replays*. However, an actual implementation is more robust than the one shown here. Lines 11 – 30 (process 1) and lines 19 onwards (process 2) show the message transmission phase. While the rest of the listings show connection-tear down phase.

Next we showcase the outline of SMTP Client (in process 2) that initiates and drives a session with SMTP Server.

PROCESS 2: SMTP Client

```
1: while true do
```

```
2:  Wait to receive message from clients; // Message Queue initialization. (Network
    service listening on port 25)
3:  state = not connected;
4:  **if** message received **then**
5:    append to message queue;
6:  **end if**
7:  **while** message queue not empty **do**
8:    message = message queue.next();
9:    **if** state == not connected **then**
10:     state = connecting;
11:     Send HELO / EHLO to SMTP Server;
12:   **end if**
13:   Wait for response;
14:   **if** response == (250 OK) **then**
15:     state = send sender;
16:   **else**
17:     break;
18:   **end if**
19:   **if** state == send sender **then**
20:     Send MAIL FROM: sender@domain.com;
21:   **end if**
22:   Wait for response;
23:   **if** response == (250 OK) **then**
24:     state = send recipient;
```

```
25: else
26:   remain in line 16;
27: end if
28: if state == send recipient then
29:   Send RCPT TO: recipient@recipient.com;
30: end if
31: Wait for response;
32: if response == (250 OK) then
33:   Send DATA;
34: else
35:   remain in line 21;
36: end if
37: Wait for response;
38: if response == (354 ENTER MAIL) then
39:   input message body;
40:   state = data;
41: else
42:   remain in line 25;
43: end if
44: if message body.end() then
45:   Send CRLF;
46: end if
47: Wait for response;
48: if response == (250 OK) then
```

```
49:         go to line 5;
50:       else
51:         append message in message queue;
52:       end if
53:     end while
54:   end while
```

Mail Transactions

In addition to a consistent view of messages transmitted in a session (as outlined above), SMTP transmissions require basic recovery procedures from transmission failures. Next, we list the two Use Cases enabled by the SMTP.

Use Case 1 Best effort transmission of a text message from a sender (the principal actor) to a recipient (the secondary actor) through intermediate mail servers (auxiliary actors).

Use Case 2 Reporting transmission failure to the sender.

A message transmission - composed of three logical steps, as discussed above – is considered complete only if the message is routed to the recipient's mailbox. The steps are: from the sender to its MTA; from sender's MTA to recipients' MTA; and finally from recipient's MTA to the recipient's mail box. Transmissions that include more than three hops can be logically treated as three hops by grouping SMTP sessions within sender and recipient domain as local delivery actions. Message transmission may fail due to many failures, like unavailability of the recipient's mailbox, or unavailable disc space, or network partitions, *etc.*; upon which the first point of failure detection is expected to generate an error message for the sender informing him or her about the failure. Note that some failures may be detected during the hop from sender's MTA to the recipient's MTA, in which case the sender's MTA detects the error. If failure, like, rejection of message by recipient's policy, is discovered after message data transmission or session termination, the recipient's MTA is considered the error detection point.

Specialization of Use Cases

Above two use cases can be specialized for a variety of message types and properties of transmission channels. Standard use cases supported by SMTP implementations are:

- Best-effort transmission of enhanced content including text and MIME messages [14].
- Enforcing security mechanisms such as transmitting authenticated message and using encrypted channel.
- Best-effort transmission of message acknowledgements.

Best-effort relates to asynchronous transfer of messages across hosts on the internet, because recipient process may not be active when the sender process contacts them. In addition, SMTP extensions [19, 20] include commands and replies for source authentication and negotiations for establishing a secure channel for synchronous transmission. Finally, SMTP also facilitates delivery receipts in the form of another email. These Use Cases are supported by functionality built into communicating server processes.

SMTP Extension Model

In order to adapt to changes in the environment and technology and add new functionalities, new extensions to the existing SMTP specifications were introduced. But, because of the large number of existing SMTP implementations, changes had to be incremental for backward compatibility and ease of adoption. Consequently, a service extension model was introduced with provisions for modifying the SMTP specifications such that *updated* Clients and Servers discover each other and transmit messages using new extensions. For transmissions with older conforming implementations, an extended Client or Server would revert back to the default protocol. This model of extending the specifications is called the SMTP extension model and new services added to the specification are called SMTP service extensions (like SMTP AUTH [19], STARTTLS [20], *etc.*). We make use of this extension model later to incorporate our solutions to the previously stated problems. In spite of the availability of many extensions SMTP infrastructures are susceptible to many misuses, discussed in next section.

2.3.3 Misuse Cases

SMTP delivery is subject to many misuse cases, including loss of privacy and integrity of content, receiving unsolicited commercial email (spam), email bombs [9], *etc.* Of these misuses the main misuses that our design prevents are:

1. **Violating integrity and leaking or altering content** : *Allowing unintended mal-actors to read message contents and alter them.* Even though service extensions [20] exist that allow mail servers to encrypt data sent over un-trusted IP network, in reality, these are seldom used. This is because both the sending

and receiving parties must agree to set up a secure channel. However, because this requirement is optional, any one party disagreeing leads to transmission of messages in plain text, thereby permitting the first misuse case to occur. In our approach, we allow senders and recipients to mandate the use of secure channels for communication in policies that are evaluated before, during and after message transmissions.

2. **Impersonating senders** : *Allowing mal-actors to impersonate others in a mail message.* As in the above case, although the SMTP AUTH extension [19] is available to mail servers for authenticated communications, it has not helped in preventing sender impersonations. There are several reasons for this deficiency. First, email identities are cheap to come by [21] – thereby making it almost impossible to ensure that communications involve known entities. Secondly, use of SMTP AUTH requires prior exchange of secrets, which is not possible in all implementations - thereby becoming a victim of the second misuse through sender-address spoofing.

3. **Email bombs** : *This is a variation of DoS attack on email networks, where mail servers receive large number of messages, leading to denial of email service.* Just like other *open* services, email servers and email mailboxes are susceptible to denial of service attacks. This is because mail servers cannot distinguish between a *good* message and a *bad* message on their own. By the time human input to prune messages arrives, it's too late for any recovery to be possible.

4. **Receiving undesirable email (spam)** : *Allowing undesirable email to reach recipients' mailbox.* Current SMTP-like server designs result in being subjected

to this misuse case, where lack of recipient control over message delivery results in receiving unwanted message in the mailbox. In other words, the protocol is heavily biased towards senders and allows them to send any number of messages any time to any recipient in the world, a fact exploited by *bulk email senders* to send spam.

Recent attention to spam has resulted in some proposals to add automated recipient controls to the message flow pipeline including, providing feedback about rejected messages [12]. A drawback of delivery controls is the inadvertent disclosure of acceptance criteria, that can now be used to defeat its purpose [13]. We also add this misuse case to the list of standard misuse cases and propose a solution in Chapter 6

2.4 Chapter conclusion

In this chapter we reviewed the current email delivery mechanisms – internet messages, email delivery protocol and the process of message delivery. We reviewed the basic Use Cases and the Misuse Cases that the current infrastructure is susceptible to. In the following chapters we introduce, develop and design our solution to prevent these Misuse Cases.

Chapter 3: Email Control Policy Architecture

3.1 Introduction

While numerous proposals have been made to address the shortcomings of email, few have seriously considered the problem from the foundational level of the SMTP protocol, which is precisely the aim of the work proposed here. To offset the bias towards senders, we propose to empower each principal with partial control of the email pipe. At any hop, the recipient of the hop can choose to drop, delay or transmit the message through increased control. My proposal allows principals to flexibly express their interests in policies, which govern their decisions during message transmission. ESPs are well positioned to provide evaluations of particular messages, sender authentication, virus-scanning and their inclusion is necessary to model proposed economic and reputation based approaches like [3, 22, 23, 24], *etc.* Primary principals provide the raw data that these servers use to make their evaluations. Adjudication services provide a third party mechanism for satisfying obligations that may be incurred as the result of delivering a mail. For example, a sender or a senders ESP may post a small bond as a guarantee for recipient satisfaction and seizure of a bond may require adjudication by an escrow service. It is important to note that the interests of the ESPs are separate from the interests of their clients. For example, a sending ESP with a reputation for harboring misbehaving senders might lose well-behaved customers if the Internet community as a whole treats all mail from the ESP with suspicion.

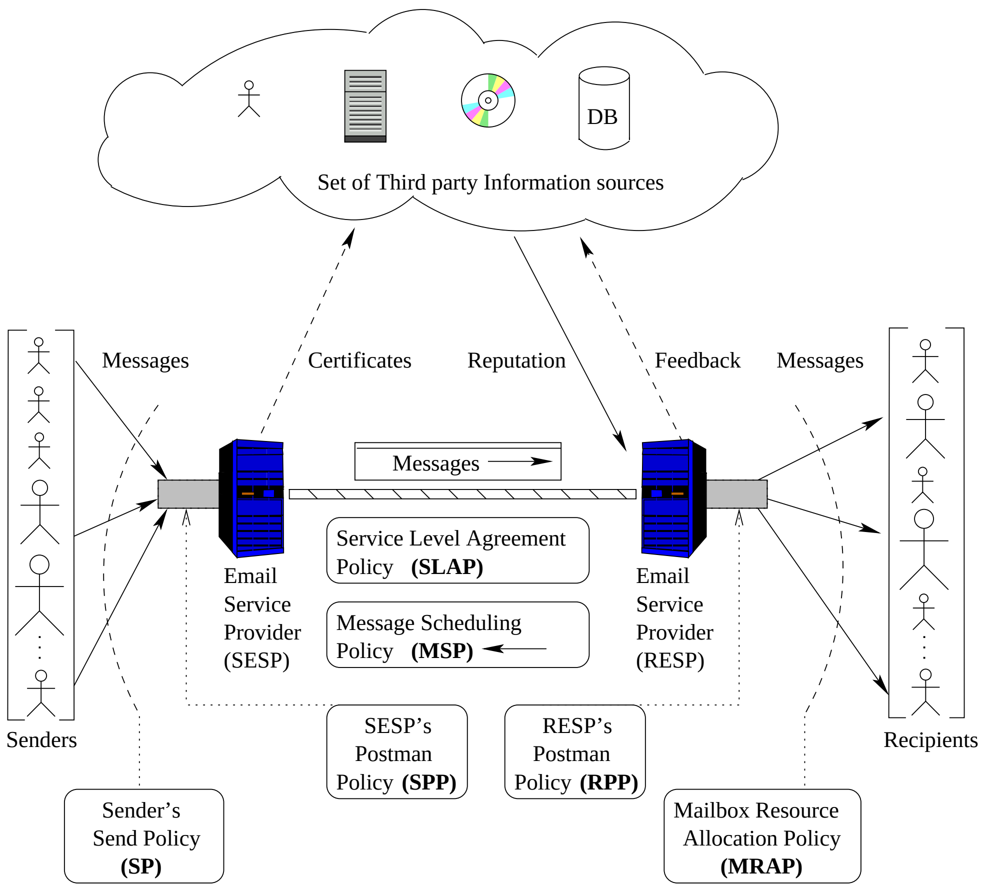
DB
Set of Third party Information sources
Messages
Certificates
Reputation
Feedback
Messages
Messages
Service Level Agreement Policy (SLAP)
Message Scheduling Policy (MSP)
Email Service Provider (SESP)
Email Service Provider (RESP)
Senders
Recipients
SESP's Postman Policy (SPP)
RESP's Postman Policy (RPP)
Sender's Send Policy (SP)
Mailbox Resource Allocation Policy (MRAP)

Figure 3.1: Principals and Policies in a Policy-mediated Email Pipe

Figure 3.1 illustrates the principals and proposed policies in a trustworthy email pipe, where messages are flowing from the left to the right. There are four principals in an email exchange. In addition to the sender and the recipient, the remaining two principals are the senders Email Service Provider (ESP), and the recipients ESP. In current practice ESPs help email service scale to the vast numbers of senders and recipients on the Internet. Each of the policies and acronyms introduced in figure 3.1 are discussed next.

There are six types of policies identified in figure 3.1 and summarized in table 3.1. These policies are expected to support the technical requirements outlined in the previous section. Here we briefly argue how enforcing these policies can prevent described Misuse Cases and facilitate all Use Cases. Following a messages journey from source to the destination, we come across various policies that can affect message delivery. At each decision point, we allow the principal to delay messages. This is to support proposals like [25, 26, 27, 28], *etc.*, which aim to slow down the rate of unwanted messages reaching the destination. By allowing messages to be dropped closer to the source, we achieve better resource utilization. For example, consider a scenario where all emails must be stamped [29, 28, 24] to get delivered. With SPP policy an SESP can check for postage and reject all unsatisfactory messages, and hence recover bandwidth that would have been eventually lost. SLAP policies are designed to enforce quality of service decisions, possibly to slow down or contain abusers as proposed in [25, 28] or outright refuse connections, and consequently can address Misuse Case 3, *i.e.*, preventing email bombs. In addition, SLAP can use other novel techniques like connections based on credentials or certificates of good behavior which do not exist yet, thereby preventing Misuse Case 1 (violating integrity and altering content) and Misuse Case 2 (Impersonating senders). MSP policy can flexibly express a host of requirements like invoking collaborative spam filters [30], virus scans and other collaborative anti-spam techniques. An MRAP on the other hand can invoke specialized rules like bond thresholds [3], blacklists, whitelists, or personalized spam filters [31]. In other words, MSP and MRAP policies prevent against Misuse Case 4 (Receiving undesirable messages). RPP allows prioritized delivery for RESP to handle requirements like prioritizing corporate email messages over personal messages, advertisements or virus-infected messages. The Sender's Send Policy (SP) is included to support and extend regulation techniques like [32], where senders mandate their

ESPs to repair rejected messages automatically. Because each principal can express their interests in a flexible manner involving all moving parts in the system, we claim that our policies are adequate in expressing most of the desirable email policies.

Table 3.1: Control achieved through policies

Policy	**Author**	**Provides/Expresses**
SP	Sender	Instructions for refining messages
SPP	SESP	Egress filtering, refining messages, effecting delays
SLAP	RESP	Quality of service to SESP, connection filtering, delays
MSP	RESP	Message acceptance criteria, refining suggestions, delays
RPP	RESP	Ingress filtering, message delaying, prioritization
MRAP	Recipient	Individual acceptance criteria, message delaying

A key question is whether the policies outlined in the architecture can adequately express the legitimate interests of the principals. We propose to approach this issue by determining whether the services offered by other proposed extensions to email fit under the umbrella provided by the proposed architecture. For example, a recipient ESP might require Turing Tests for certain classes of mail, but these might be best administered by the sending ESP during the evaluation of the SPP. In this example, the proposed architecture needs to accommodate the efficient flow of one principals requirements to software agents operating on behalf of another principal. Next we discuss each policy in a little more detail.

3.2 Policy architecture

3.2.1 Sender's Send Policy(SP)

Sender's Send policy (SP) is a collection of preferences at the sender to deal with undelivered messages. That is, senders can mandate their ESP to add 'attributes' to undelivered messages such that they are able to satisfy recipient's message acceptance policies (MSP and MRAP). Consider a scenario where the target recipient accepts messages only if they are appropriately bonded [3]. If an important message from sender is returned by recipient due to inadequate bond value, the sender may direct its ESP to automatically increase the bond value, up to a maximum, to get the message accepted. By specifying preferences for expected type of rejections, a sender can hope to get important messages revised and delivered, automatically.

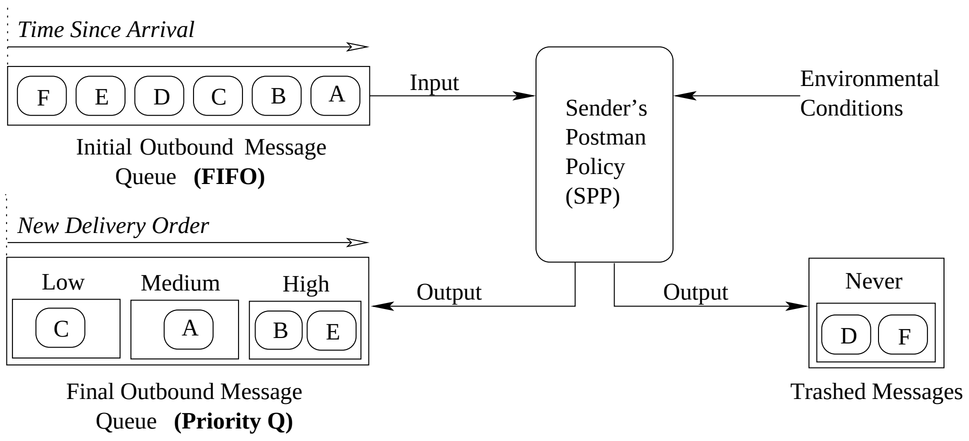

Figure 3.2: SESP's Postman Policy (SPP)

3.2.2 SESPs Postman Policy (SPP)

SPP reflects the interests of the SESP and specifies the SESPs mail transmission requirements, such as giving priority to delivery of urgent messages from preferred senders or to preferred email domains. Requirements also include handling virus outbreaks appropriately; other circumstances may also justify application of filters to outgoing mail, eg, [33] makes a case for applying extrusion filters to outgoing mail. The SPP also controls the addition to the message of sender characterizations signed by the SESP to support two level authentication proposals like [34] and hence plays a secondary role of an annotator and enabler for sender authentication. For example, verified signatures can be added to senders emails. It also acts as a negotiator when messages sent from its domain get rejected. Inputs to SPP include: sender attributes, the number of enqueued messages from a given sender, sender email subdomain, recipient email domain, message scan results, and the state of the environment.

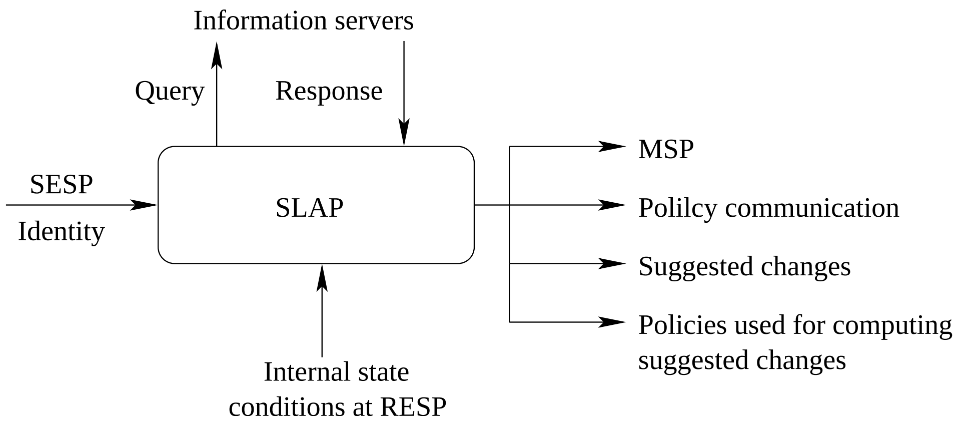

Figure 3.3: Service Level Agreement Policy (SLAP)

3.2.3 Service Level Agreement Policy (SLAP)

Connection-level actions of an RESP are governed by its SLAP. The inputs to SLAP include identity of the SESP, environmental conditions at the RESP, and the reputation of the SESP. This reputation is presumed to summarize past experiences RESP has had in dealing with the SESP or experiences of third parties [35, 36]. The essential requirement is that the reputation service characterizes the past behavior of SESPs so that the SLAP can give preferential treatment to SESPs with long records of favorable behavior, or no record of bad behavior. Relevant environmental conditions might include work load at the RESP or virus alerts, etc. Together these inputs characterize the current SMTP session; based on them, the SLAP specifies gross transmission scheduling. Candidate specifications include: send them now ("as is"), delay them ("later"), and do not accept messages from this SESP ("never"). In the most general case, however, this specification is itself a policy, called a Message Scheduling Policy (MSP), discussed further in the next section, that considers each message in turn, and determines whether and when it should be transmitted. SLAPs are not necessarily shared between principals, so their format is potentially organization-specific.

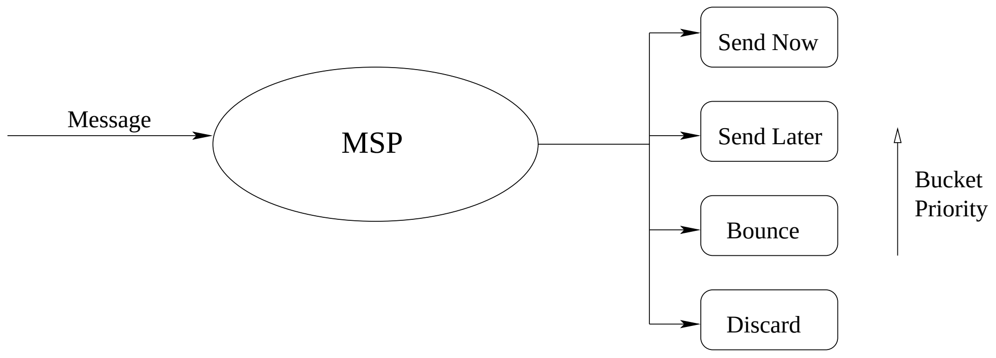

Figure 3.4: Message Scheduling Policy (MSP)

3.2.4 Message Scheduling Policy (MSP)

MSP is applied to each message awaiting transmission to determine whether and when it should be sent from the SESP to the RESP. However, stopping unfit messages is not its sole function, instead, it acts as an facilitator for delivery of messages that either fit recipients preferences or can be improved by minor mutations, to satisfy the recipients policy. The scheduling of a message depends on the characteristics of the message and its sender, and any other criteria of interest to the recipients, like reputations, bonds etc. We model each of the criterion as a predicate that takes a message and returns true if the message, its sender, or the SESP involved satisfies the predicated characteristic; otherwise it returns false. Characteristics of interest can include any the following:

- sender characterization: sender authentication and its means, reputation of SESP, rate of sender activity etc.;
- Message characteristics: Content classification [34], message reputation [23], monetary guarantees [3], score on spam filter [37], *etc.*

MSPs are are designed to flow. Its two types are: portable and local. The difference lies in where the policy is evaluated. A portable MSP is transmitted from the RESP to the SESP, which is trusted to apply the MSP to determine which messages to transmit. A local MSP can be used when the recipient ESP wishes to apply an MSP to the messages transmitted, but does not wish to entrust the SESP with this task. In a local MSP application, the SESP transmits messages headers, which the RESP then uses to evaluate the MSP on each waiting message. The RESP then provides the SESP with instructions as to whether and when to transmit each of the waiting messages. The format of local MSP is ESP-determined, whereas, because it is evaluated remotely, the format of portable MSP must be universal. In either case, the RESP has the capability to check whether its MSP was adhered to by the SESP, and can send this information to reputation repositories for later use. As an example consider the policy which states that messages that have been sent by authenticated senders, and pass extrusion spam filters or virus scans, be allowed transmission. It can also say that if the sender cannot be authenticated, then a reliable bond value with a minimum monetary value is required for acceptance. As a result only those messages that satisfy the policy are accepted by the RESP.

3.2.5 RESPs Postman Policy (RPP)

RPP is analogous to SPP, though it reflects the interests of the RESP. It defines priorities for delivery of inbound messages to mailboxes depending on characteristics of the sender, the recipient, and the message itself. This enables messages to be reordered so as to ensure that urgent or high-quality messages are delivered in a timely way, even when many messages compete for delivery. It may also specify message processing requirements such as virus scanning, and actions triggered, such as notification to interested principals. For example, an ESP can define an RPP that mandates preferential delivery of urgent messages from business partners, messages to recipients that are preferred customers, bonded messages, and messages from highly trusted SESPs that have vouched for them as being of high quality.

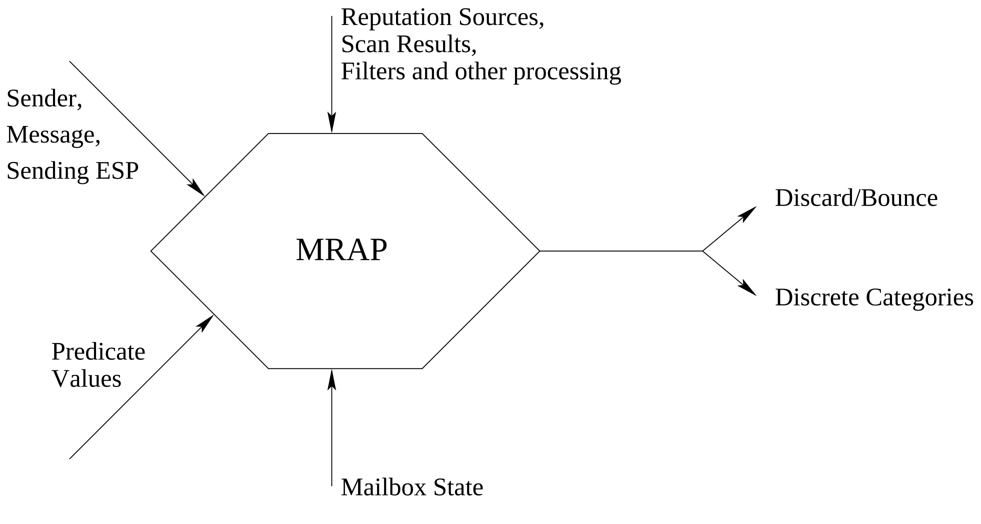

Figure 3.5: Mailbox Resource Allocation Policy (MRAP)

3.2.6 Mailbox Resource Allocation Policy (MRAP)

The Mailbox Resource Allocation Policy (MRAP) is specified by mail recipients and controls the utilization of their mailbox folders. User mail-reception preferences are captured and converted to MRAPs, which are evaluated by the recipient ESP as part of the process of delivering a message to the user. The input to MRAP includes: the complete message (including content); sender identity, characteristics, and reputation; SESP identities and reputations; message attributes; mailbox state, *etc.* The result of MRAP evaluation is either to discard or bounce the message, in which cases the message is not delivered to the mailbox, or to assign a discrete category (i.e., a mailbox folder) to route the message to. MRAPs can also *flow*, like the MSP policies, however, MRAP communication is subject to tighter dissemination controls. This is because MRAPs communications pose a greater risk in leaking private information.

3.2.7 Message Negotiation

Consider a scenario where a sender's message is rejected due to a lack of monetary guarantee [3]. If a sender really wants her message delivered, and is prepared to make amends to it, can the delivery mechanism facilitate a change to her message and get it delivered? Through the negotiation mechanism we provide such a facility and hence support and improve upon similar proposals [38]. The sender clearly has the largest interest in making messages meet their recipient's policies. However, it is entirely reasonable that a sender's service provider would, as a result of the provider's contractual relationship with the sender, be prepared to act as the sender's agent in supplying the repairs, automatically. Given a message and a corresponding policy that rejects that message in its current form, we would like to be able to find the most appropriate fix, if any, to enable that message to satisfy the policy. To the extent allowed by privacy concerns, policies should be made available as close to the sender as possible to maximize the chances of a successful repair. From a policy standpoint, such negotiations take place between MSP, MRAP and SPP policies; though each depends upon the end users preference policies to complete such a negotiation.

To implement this scheme in the extended SMTP protocol (e-SMTP), we need to devise a mechanism for simple negotiations and define the semantics of message rejection. Following chapters present a constraint logic programming based approach to achieve the former goal. In essence, this is made possible by making a small change in the semantics of *accept or reject decisions*. In the prior policy evaluations only two possible outcomes are supported: *Acceptance* or *Rejection* of a mail message. However, to allow message negotiation, a reject is interpreted in one of the two ways: *a temporary reject* characterized by a set of *fixes* that may change the decision into an accept; and *a permanent reject* characterized by an empty set (of fixes) that represents the fact that the message cannot be *repaired*. Also, because the accept decision is valid only for the current hop, it does not signify that the message will be delivered to the recipient, and in this sense, its only a *temporary accept*. A *final accept* is authorized only by the MRAP policy to the messages that are able to make it to that hop. In summary, by allowing the possibility of temporary accept and reject decisions, we introduce the possibility of message negotiation. From an implementation standpoint, the key issue for negotiation mechanism is how to identify and evaluate efficiently potential message modifications. We return to this in the next chapter.

3.2.8 Privacy

One consideration that acts in opposition to the desire to negotiate in fixing messages or making policy known upstream is the sensitivity of policy content. Information used by the recipient or the RESP to decide whether a message is likely spam may need to be protected from the sender or the SESP. Such sensitive information could include, for example, blacklists, whitelists, and filtering rules. These examples can be considered part of any policy that uses them. Private policy content creates a trade-off between making policy content available upstream so as to enable unwanted messages to be dropped or fixed as early as possible, and the need to protect sensitive policy content. One measure that we support will transform the actual policy to derive a related policy that makes no use of the sensitive portions of the original policy. For example, a sensitive MSP can be transformed to obtain a sanitized MSP that can be distributed to SESPs to enable them to filter the email they send to the RESP. The sanitized MSP should be related to the original in the following sense: it should accept any message that the original would accept under some definition of the private portion of the policy. In this way, the sanitized MSP does as much filtering of messages as is possible without knowing the private portion of the original policy.

Using one strategy that we plan to support, the RESP accepts messages transferred to it by the SESP provided they satisfy the sanitized MSP. Then, after the message is transferred, the RESP drops messages that do not satisfy the original MSP. In some cases, we expect this to be considered adequate protection for the private portions of the policy. However, the security guaranteed by dropping messages will not be strong enough for some contexts.

Because the mail system is highly automated, there is a potential for a great deal of information about policy content to be leaked without direct transmission of the policy's representation. An attacker can simply send a large number of email messages and observe the results in an effort to discern the policy. Therefore it is particularly important that the amount of feedback available to the attacker be limited. Here we are not so concerned with feedback that can be obtained by out of band channels, like a recipient phoning a sender when a message is received. There is little we can hope to do about such signals and their bandwidth is typically relatively low. What we do aim to provide is a guarantee that the system itself will not signal whether a message is accepted by a policy when that acceptance depends on private information in the policy. In-band signals of concern includes email system events the sender can observe, such as the seizure of a bond or the change in a public reputation value. Thus, while previous work on the protection of sensitive policy content such as the UniPro system [39] has tended to focus on controlling access to the direct representation of the policy, we take an information-flow approach by ensuring that there is no dependence of the systems observable behavior on private policy content.

3.3 Policy Composition and Implementation

The descriptions of policies in the previous section yield specifications for how best to serve the interests of the various principals in an email exchange, but they say nothing about how such policies can be implemented in an incremental way on top of the current infrastructure for email - namely the SMTP. Such an implementation poses many challenges in the heterogeneous world of the Internet. Figure 3.6 outlines our basic approach to address this ambitious mapping. For simplicity, third parties

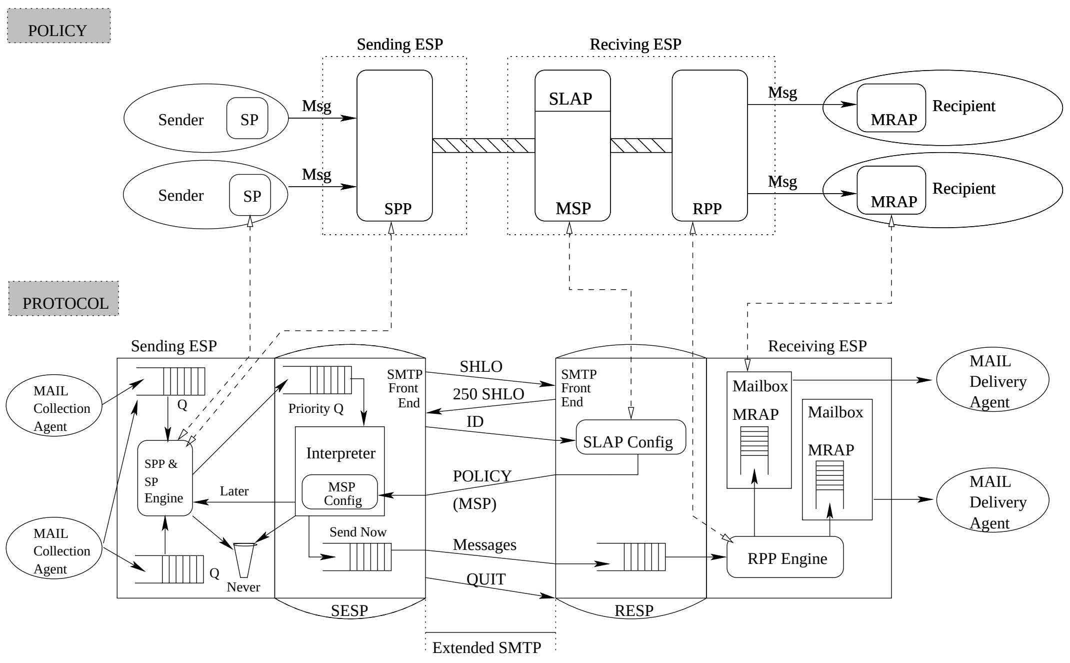

Figure 3.6: Policy-Mediated Email Pipe

such as reputation servers are not shown in the diagram.

The most critical aspect shown in the diagram is a replacement for SMTP, and is labeled “Extended SMTP” in the diagram. The partial “ladder diagram” for extended SMTP shows, at a very high level, the SESP authenticating itself to the RESP, accepting an MSP Policy from the receiving MTA, and then sending, delaying, or deleting messages as specified by the MSP. (Alternatively, the SESP could choose to violate the MSP, which could damage its reputation.) Details of the proposed extension to the ladder diagram will be presented later.

Figure 3.6 also shows the enforcement points of the remaining five policies. At the policy level, the left to right axis shows the successive applications of policies to messages. At the protocol level, the left to right axis shows the successive entities responsible for implementing the policies. In particular, the Sending Postman Policy

(SPP) and Sender's Policy (SP) are implemented with an engine at the SESP. We propose to encode the SLAP of the RESP - the policy that governs how a RESP treats a particular SESP, in a SLAP configuration that efficiently provides an appropriate MSP to the SESP. Because this function occurs during an execution of Extended SMTP, there are severe performance constraints on its implementation.

Postman functions at the RESP, including virus scanning and other hygiene functions can be carried out offline from the Extended SMTP protocol itself, thereby simplifying the performance demands. The output of the RPP evaluation is a priority queue specifying the message delivery order. Unlike the current SMTP, the RPP queue may intentionally delay certain messages for the purpose of allowing reputation sources like [23, 40] to weigh in before scheduling the messages for delivery. Finally, we envision most MRAP policies being cached at the RESP. As an aid for end users, one strategy to author MRAP policies could be to profile the users' preferences in messages that gets automatically converted to an MRAP policies for use in future message transmissions. More sophisticated MRAP policies specifically reflect specific end user preferences and may be reserved for more advanced users.

3.4 Chapter conclusion

Designing adequately expressive policy languages, with adequate expressive power with respect to authorization, liveness, and fairness, suitable for email application is the primary research issue of this disseration. These features must be balanced against the requirement that they be implemented efficiently. In the case of portable MSP, we need to assess the efficiency and usability issues associated with designs: one alternative would transmit actual policy rules; another transmits references to standard sets of pre-defined policy rules. Some combination of the two may also be appropriate.

A semantics of policies and their compositions should define the behavior of the pipe at a suitable level of abstraction. A precise semantics is necessary so that we can articulate the correctness requirements for the distributed implementation of the various policies and the composition of their effects. Specifically, each policy must map directly to one or more implementation components at the protocol level that enforce it. In addition, the effect on message transmission at the policy level of figure 3.6 should match the effect at the protocol level.

In this chapter, we elucidate several abstract requirements and high level concerns while solving the delivery decision problem. Because parameterizing the behavior of the network with policies, email transmission becomes more complex and must be carefully engineered. Next several chapters are devoted to each of these concerns and collectively solve the problems introduced in this thesis till now.

Chapter 4: The Formal Model of Policies

4.1 Introduction

Sender bias in SMTP protocol is usually offset by using several types of email control mechanisms at the receiving end. However, automatic enforcement of restrictions on transmission of email messages presents a dilemma: the enforcer faces a risk of screening out desirable messages. This is partly because unwanted messages, also known as spam, are increasingly being camouflaged as acceptable messages, thereby making filtration difficult on the basis of content. Most of us are more tolerant of receiving unwanted messages than of losing desired ones, forcing us to make the acceptance criteria more permissive. Techniques for expressing desirability of messages abound, and we discuss prominent ones in the related work section. With these techniques, recipients and other principals, such as recipients' email-service providers, can express diverse criteria to define message acceptability. However, this is of little help to the sender or their service providers, who may be unaware of how documentation can be added to a message to make it acceptable downstream. We aim to address this shortcoming.

In the simplest solution, downstream principals provide the sender with some sort of feedback about message acceptance requirements. Consider, for example, a policy that allows messages ranked as likely spam (by a Bayesian filter) may nonetheless be delivered, provided a reliable third party authenticates the sender. By providing this feedback, say, as a policy advertisement or as a suggested change to a rejected message, senders can be given the opportunity to adequately document their messages so they are delivered. We call this process, of providing relevant feedback to senders and their agents for modifying rejected messages and the subsequent modification of rejected messages before retransmission, *message refinement*.

The mushrooming of techniques purporting to control the delivery of unwanted messages presents both opportunities and challenges. Their increasing number bears evidence to the fact that there are many different aspects to the email control problem. Yet each individual solution tends to address only a part of the problem, like, Bayesian filters [30] only consider keywords in the message content, while captcha tests [4] check for human initiation only, *etc.* The current approach of designing a new system for every new scheme is clearly impractical and unscalable. Moreover, there is a lack of autonomy in combining different techniques to suit the needs of a particular site. A basic challenge in this area is to design a general purpose system to combine, support, and enforce any and all mechanisms within the SMTP [1] framework, the default mail delivery protocol. Such a system is also a prerequisite to a scheme that provides a generic way to help refine messages.

We examine the policy framework discussed in previous chapter, designed to address this challenge. This flexible framework for specifying message acceptance criteria enables most existing email control techniques to be used in concert. The framework allows each principal involved in message transmission to express policies, be they for relaying messages to the recipient, for storing them in the mailbox, or expressing 'refinement' preferences. Based on the message attributes and content, these policies determine whether a message is delivered, dropped, or rejected, and in the latter case, whether changes are suggested to more adequately document the message.

For example, an acceptance policy can state that a message must satisfy one of the following conditions to be acceptable: ‘has a low score on a Bayesian filter’; ‘has evidence of human initiation’; ‘the message sender is trustworthy’. In this case, a message is authorized for delivery to its destination only if it is accompanied by documentation that shows it satisfies one of the three criteria. Policies that use conjunctions of requirements can also be expressed. For example, (`mail-server-authenticated` `=` *true*) $\wedge$ (`reputation` $\geq$ 70%) $\wedge$ (`bond` $\geq$ 5) requires simultaneous satisfaction of requirements involving multiple email-control mechanisms. In this case, the sending email server must be authenticated and have a reputation value exceeding 70%, and the message must be bonded, which enables the recipient to be financially compensated after accepting a message if he subsequently so chooses. Each mechanism will provide its own basis of verifying associated message documentation. This could either be, for example, a distributed reputation system [41], a monetary guarantee [3] or a digital certificate [5].

In our approach, feedback about rejected messages consists of suggestions about how to correct the deficiencies in a rejected message so that the acceptance criteria may be satisfied. Because email messages are transmitted in several hops, messages can be rejected at any hop. (Indeed one of goal of our research is to enable undesirable messages to be recognized as early in the transmission process as possible [42].) Feedback is generated at the hop at which the message is rejected, and is communicated upstream. Details of this scheme are presented later in section § 4.3.

Confirming that an email address is valid by providing feedback for messages addressed to it can result in the recipient being targeted with more unwanted messages.

Additionally, while it is our aim is to make it possible to specify policies that cannot be satisfied by tricking the system, in practice providing feedback to malicious senders may help them craft undesirable messages with better attributes, so that they will be accepted. By putting the decision whether or not feedback is provided under policy control, these risks can be minimized. In essence, policies can assess the trustworthiness of transmitters, and decide how much information, if any, should be disclosed to them by way of feedback.

Here we focus on policies that determine which messages are acceptable and how unacceptable messages could be better documented so as to make them acceptable. In the approach we investigate, policies are constraint logic programs (CLP). The use of constraints is particularly helpful when providing feedback to senders and their agents about messages that are rejected.

To provide a uniform view within the policy, we model several forms of message documentation as headers on the message itself. These values must be made available within the program that is the policy. In addition, system environment parameters, such as system load, time of day, and security-alert status must be available.

The most straightforward means of making such values available is through parameters passed in through queries to the system. In this case, these values are taken as parameters by the predicates that define the interface to the policy used by the mail system. However, fully parameterizing these predicates with message header and content, as well as system parameters, raises concerns about tractability of evaluation, as well as privacy of system parameters. For example, a Bayesian spam classifier assessing message content clearly depends on the message content for evaluation. We model mechanisms such as this spam recognizer as predicates in our policies. In the fully parameterized approach, message content is a parameter to the spam-recognizer predicate. In a top-down program evaluation, such as is used by Prolog, such a predicate needs to be evaluated only for input parameters that are available when the use of the predicate is reached during evaluation. However, top-down evaluation can perform so many redundant steps that it becomes intractable. In a bottom-up evaluation, these redundant steps can be avoided. Yet in this case all true atoms constructed with the spam-recognizer predicate must be calculated, which, given the number of possible messages, is clearly impractical.

While it may be possible to address these concerns in other ways[1], the approach we consider here models the assignment of values by defining special-purpose predicates whose models correspond to the current message headers and content, and the current system-parameter values. These values are then referenced in the policy definition by using these special-purpose predicates. Thus, in the approach considered here, evaluation of these predicate instances are with respect to attribute values extracted from email headers. In other words, each message and system state determines a new evaluation of the predicates. This approach leads to several interesting and attractive results, which we develop here. It remains a matter of future work to complete an assessment of whether this approach is the most attractive possible.

The use of policies for email control requires changes to the SMTP protocol. In particular, email-control techniques that require various kinds of challenges to be presented to a sender [26, 27, 4] are difficult to support in the current SMTP protocol. As argued earlier, changes are essential to obviate the need for designing

[1]For example, it may be possible to use a hybrid approach to evaluation, which combines features of top-down and bottom-up evaluation [43].

many protocols layered over SMTP for supporting different mechanisms. We suggest appropriate extensions to the current protocol to satisfy such requirements. Through our extensions we also achieve the capability of *advertising* policies upstream, which can further help the senders and their agents in screening messages for transmission. More details are presented later in chapter 7.

In the rest of this chapter we present the following topics. We begin with a flexible policy language for expressing email control criteria involving a majority of the existing email control mechanisms. We identify a constraint domain which supports requirements involving existing control mechanisms. Next, we provide a logical evaluation procedure for making email-delivery related decisions. We present what is to the best of our knowledge the first tractability result for evaluation of normal CLP programs in the identified constraint domain. Finally, we present a novel scheme for refining messages to meet precise acceptance criteria. We design a PTIME algorithm to find a suitable refinement to rejected messages, supporting our argument that it is practical to be implemented efficiently in user's email agents.

4.2 The Formal Model

Policies specify acceptance conditions on message headers and content. In § 4.2.1 we provide the syntax of the MSP and MRAP policies, as their format is required to be universal [42] to permit policy advertisement. Formats of other policies like SLAP are domain dependent. In § 5.2.2 we introduce our scheme for policy application and message refinement for rejected messages. Policies in our model are constraint logic programs [44]. Next we provide the syntax of policies that can control and refine messages.

4.2.1 Syntax

The policy language we use is presented next.

Definition 1 (Constraint domain)**.** *Constraint domain we consider is the finite integer domain that supports standard interpretations of the following symbols:* $=$, $\neq$, $\leq$, $\geq$. *We denote our constraint domain by* $\mathcal{R}$. *In addition, the constraint domain supports interpretation of minimize(G, E) predicates defined in [**?**].*

We use elements of $\mathcal{R}$ to implicitly encode all alphanumeric constants like sender@abc.com, recipient@xyz.com, *etc.*

Definition 2 (Predicates)**.** *Predicate symbols not interpreted by the constraint domain are partitioned into the following sets:*

Special Predicates (SP) *Nullary predicate symbols accept, allow and disallow, and 8-ary predicates revise, rAllow, and rDisallow.*

Environmental Predicates (EP) *Unary predicates -* $atrb_{From}, \ldots, atrb_{Sesp}$, $syst_{Load}$, $syst_{Time}$, $prim_{my\text{-}crm}$, $prim_{virusScan}$, $prim_{RepService}$, $prim_{lumosRep}$, $prim_{lumosPriority}$ *and* $prim_{priorityServer}$.

User Defined Predicates (UP) *Such as whitelist, blacklist, partner, etc. (We provide examples of how UP predicates are used later in this chapter).*

The arity of revise, rAllow and rDisallow corresponds to the number of headers supported (6) and two system environment variables. Predicates and their meanings are summarized in table 4.1

Definition 3 (Term)**.** *A* term *is a variable or a constant.*

Table 4.1: Predicates

Predicate/ Arity	**Description**
$\text{atrb}_X/1$	Various environmental predicates created from message headers
$\text{syst}_Y/1$	Various environmental predicates created from current system state
$\text{prim}_Z/1$	Various environmental predicates created from the evaluation of primitive mechanisms
whitelist/1	User defined predicate indicating a trusted sender
blacklist/1	User defined predicate indicating a 'blocked' sender
partner/1	User defined predicate indicating a business partner
allow/0	User defined predicate for acceptance condition
disallow/0	User defined predicate for rejection conditions
accept/0	System defined predicate that determines the acceptance of a message
rAllow/8	User defined predicate indicating desirable message attributes
rDisallow/8	User defined predicate indicating undesirable message attributes
revise/8	System defined predicate for computing revisions for satisfying downstream policies

Definition 4 (Atom and Literal)**.** *An* atom *is of the form $q(t_1,\ldots, t_n)$ where q is a predicate symbol or a primitive constraint and $t_1,\ldots, t_n$ are terms. A* literal *is an atom (called a* positive literal*) or its negation (called a* negative literal*).*

Definition 5 (Constrained atom)**.** *A* constrained atom *is a pair, represented as $c|A$ [45], in which c is a solvable constraint [44], A is an atom, and $fv(c) \subseteq fv(A)$, where $fv(\cdot)$ yields the set of free variables occurring in its argument. The set of constrained atoms is denoted by $\mathcal{B}$.*

Definition 6 (Clause, Fact and Rule). *A* clause *is of the form $H \leftarrow B$ where H is an atom (called the* head *of the clause), and B is a list of literals (called the* body *of the clause). A* fact *is a clause in which B is an empty list or a list of literals constructed using constraint predicates. A clause is called a* rule *otherwise.*

Definition 7 (CLP Program). *A CLP* program *is a set of clauses constructed from the terms and predicate symbols introduced above.*

Definition 8 (Query). *A* query *Q is a list of literals.*

Definition 9 (Program dependency graph). *Let P be a CLP program and V be the set of predicate symbols occurring in P. The* program dependency graph of *P is the pair $\langle V, E\rangle$, $E \subseteq V \times V$ in which, given $p, q \in V$, (p, q) is in the edge set E if and only if there is a clause $H \leftarrow B$ in P, in which H is p and there is a literal in B constructed using q. The edge (p, q) is labeled '+' (called a* positive reference*) if the literal, q, is positive, and '−' (called a* negative reference*) otherwise.*

Definition 10 (Stratified program). *A program P is* stratified *if no edge labeled '−' in the program dependency graph of P occurs in a cycle.*

The predicates of any stratified program, P, can be partitioned into disjoint sets called *strata* with the following property: there is a partial ordering of the resulting set of sets of predicates in which each clause of P that defines a predicate in any given stratum can only contain those predicates defined in the same or lower strata, to be in the body of the clause; and allows negative literals in the body that are defined in a lower strata. We abuse the terminology slightly by also using "stratum" when referring to the set of clauses of P that define predicates in a given stratum.

Policy definitions

MSP and MRAP policies could be shared among the communicating agents. We next provide their syntax. The other policies like SLAP, SPP, SP and RPP, *etc.*, are not shared among the principals and hence their syntax need not be universal, and can be implementation specific. We define the complete syntax of all policies we introduce, in the next chapter.

Definition 11 (MSP & MRAP Policy)**.** *An* MSP or an MRAP policy *is a stratified program with the following strata:*

- ***Stratum 1:*** *Definition of facts defining the predicates in EP.*
- ***Stratum 2:*** *Definitions of UP predicates.*
- ***Stratum 3:*** *Definition of SP predicates allow, disallow, rAllow and rDisallow. Negative literals are allowed in the body, however they must be constructed using Stratum 0 or Stratum 1 predicates.*
- ***Stratum 4:*** *This stratum consists of one clause: accept()* $\leftarrow$ *allow(),* $\neg$*disallow()*
- ***Stratum 5:*** *This stratum consists of one clause: revise(*$\tilde{X}$*)* $\leftarrow$ *rAllow(*$\tilde{X}$*),* $\neg$*rDisallow(*$\tilde{X}$*)*

UP predicates defined in the Stratum 2 of MSP and MRAP policies may invoke negative literals in their bodies, however, the use of negation is restricted, in that, the set of UP predicate definitions must be stratified. Also, Stratum 5 uses vectors of variables as arguments of predicates. This is to capture information on predicates that can be *refined*. For predicates in SP, the definitions of `allow` and `disallow` are written by the policy author. The definitions of `rAllow` and `rDisallow` are constructed from these automatically. The definitions of predicates in UP are also written by the policy author. The remaining two clauses of the policy are the same in all policies.

4.2.2 Encoding emails into the constraint domain

In this section we show how we encode emails into our constraint domain. We begin by encoding email headers, defined next.

Definition 12 (Headers). *Reserved constants that refer to message attributes are called* headers*.*

For simplicity, we consider only the following headers: `Mail From`, `To`, `Date`, `X-SESP`, `X-Bond` and `X-Auth`, represented by, respectively, *From*, *To*, *Date*, *Sesp*, *Bond* and *Auth*. This list can be easily extended.

We use $\mathcal{H}$ to denote the set of facts derived from message headers, like `from, to`, *etc.* We use $\mathcal{M}$ to denote the set of facts that define current state conditions and the evaluation of the primitive mechanisms, such as bayesian filters, virus-scans, *etc.* $\mathcal{H}$ and $\mathcal{M}$ define environmental predicates (EP), consequently, Stratum 1 in MSP and MRAP policies is the union of $\mathcal{H}$ and $\mathcal{M}$ and are generated automatically using the allowed EP predicates.

Just prior to policy evaluation, $\mathcal{H}$ is generated from a message by a preprocessor in the following way. Create a fact for each message header by introducing a clause, *i.e.*, $\text{atrb}_{Header_i}(\text{X}) \leftarrow c_i$, where $Header_i$ is a meta variable whose value is a header name and c_i is a primitive constraint on the variable X, constrained by the header value, for example, a bond attribute in a message, such as `X-Bond in [0, 3] USD`, is encoded as: $\text{atrb}_{Bond}(X) \leftarrow X \in [0,3]$. Note that this requires a change in how message headers are specified, however, we expect this change to affect only the X-header specification, *i.e.*, a new X-header `X-Bond:in [0,3] USD` would mean that message is bonded with value in [0,3] interval and the unit of bond is U.S. Dollars. Headers not used in the policy are ignored. For headers used in the policy, but not present in the message or present but with no values specified, policy-defined default facts are used.

$\mathcal{M}$ is generated as follows. Similar to the construction of header-facts in $\mathcal{H}$, current state conditions at the recipient side are also expressed as facts, *i.e.*, $\text{syst}_{SysVariable}(\text{X}) \leftarrow c_i$. Corresponding to each instance of a primitive mechanism is a single predicate defined by a single fact that provides the result produced by that mechanism instance on implicitly specified input. For example, if a particular CRM filter [30] (my-crm), when applied to the message, returns the score value 1, then there will be one fact in $\mathcal{M}$ of the form $\text{prim}_{my-crm}(1)$. Note that the restricted way in which primitive mechanisms are used ensures that all relevant values can be pre-computed prior to evaluation, so there is no need to "call out" to the mechanism. This enables us to use a bottom-up semantics for evaluation.

4.2.3 Semantics

We now present a fixpoint semantics for CLP programs that is PTIME complex, hence tractable, for the constraint domain we use in our email policies. It constructs the semantics as the least fixpoint of an immediate-consequence operator due to Fages [45]. As usual, the domain of this operator is the set of interpretations. An *interpretation* I is a pair $\langle I^+, I^- \rangle$, in which I^+ and I^- are disjoint sets of constrained atoms.

In the following definitions, we assume that clauses are in the following standard form: $H \leftarrow c \mid B_1, \ldots, B_m, B_{m+1}, \ldots, B_n$, where $B_1, \ldots, B_m$ are positive literals, $B_{m+1}, \ldots, B_n$ are negative literals, c is a constraint, and, except in c, no variable occurs more than once. We also assume for the present that the function $QE(\cdot)$ satisfies $QE(c) = \{c\}$ for any constraint c. In section 4.5 below, we replace this trivial definition of $QE(\cdot)$ with a more interesting definition that performs quantifier elimination.

Definition 13. (Immediate-consequence function [45])
$T_P(I)= \langle T_P^+(I), T_P^-(I)\rangle$ in which:

$T_P^+(I) = \{c'|p(X) \in \mathcal{B}$: there exist a $p(X) \leftarrow d|A_1, \ldots, A_m, \neg A_{m+1}, \ldots, \neg A_n \in P$ with local variables Y, $c_i|A_i \in I^+$ for $i \in [1, m]$ and $c_j|A_j \in I^-$ for $j \in [m+1, n]$ such that $c= \exists Y(d \wedge \bigwedge_{i=i}^{n} c_i)$ is satisfiable and $c' \in QE(c)\}$ and

$T_P^-(I) = \{c'|p(X) \in \mathcal{B}$: $p(X) \leftarrow d_k|A_{k,1}, \ldots, A_{k,m_k}, \neg A_{k,m_k+1}, \ldots, \neg A_{k,n_k}$ for every clause with head $p \in P$ and local variables Y_k, there exist $e_{k,1}|A_{k,1}, \ldots, e_{k,m_k}|A_{k,m_k} \in I^-$ and $e_{k,m_k+1}|A_{k,m_k+1}, \ldots, e_{k,n_k}|A_{k,n_k} \in I^+$, such that $c= \bigwedge_k \forall Y_k(\neg\, d_k \vee \bigvee_{i=i}^{n_k} e_{k,i})$ is satisfiable and $c' \in QE(c)\}$

Definition 14. (Ordinal powers of T_P)
$T_P \uparrow 0 = \emptyset$; $T_P \uparrow \beta+1 = T_P(T_P \uparrow \beta)$; $T_P \uparrow \alpha = \bigsqcup_{\beta<\alpha} T_P \uparrow \beta$, in which α is a limit ordinal and $\bigsqcup_{\beta<\alpha} T_P \uparrow \beta = \langle \bigcup_{\beta<\alpha}(T_P \uparrow \beta)^+, \bigcup_{\beta<\alpha}(T_P \uparrow \beta)^- \rangle$.

This construction requires of the constraint domain ($\mathcal{R}$) to be *admissible*, meaning that for any constraint (atom or constant) c, there exist $c_1, \ldots, c_n$ such that $\mathcal{R} \models$

$\neg c \leftrightarrow \bigvee_{i=1}^{n} c_i$ where c, and each c_i are primary constraints [45]. This is true of $\mathcal{R}$, because the complement of any interval is a union of intervals.

Definition 15 (Query Evaluation)**.** *Given program P and letting I=$T_P(\emptyset) \uparrow \omega$, the* answer constraint *for query Q is*

$\sigma_a = \bigvee\{\sigma$*: $Q = c|A_1,\ldots,A_m,\neg A_{m+1},\ldots,\neg A_n$., where each A_i is a clause in P, $c_u|A_u \in I^+$ for $u \in [1,m]$ and $c_v|A_v \in I^-$ for $v \in [m+1,n]$ such that $\mathcal{R} \models \sigma$, in which $\sigma = c \wedge \bigwedge_{r=1}^{n} c_r\}$*

Message delivery is determined by evaluating the query Q = *accept*(). If the answer constraint for Q is equivalent to false, the message is rejected. Otherwise, it is accepted for delivery.

Example 1. *Policy evaluation without feedback.* Consider a message with only three headers:

From: sender@abc.com (final); To: recipient@xyz.com (final); X-Auth: 'Password'(final)

Keyword 'final' in the headers indicates that the header cannot be changed. It is explained in more detail in section 4.3. Consider a simple MSP:

$$allow() \quad \leftarrow \quad atrb_{Auth}(X_{Auth}), X_{Auth} = \text{`}PKI'$$

This policy accepts only strongly authenticated messages. The fact defining $\text{atrb}_{\text{Auth}}$ that is constructed from the message is:

$$atrb_{\text{Auth}}(X_{\text{Auth}}) \quad \leftarrow \quad X_{\text{Auth}} = \text{`}Password'.$$

Since authentication provided in the message is inconsistent with that required in the policy, the accept() query fails, and the message is rejected.

Definition 16 (Projection)**.** *We denote the* projection [44] *of a constraint c onto a set of variables V by proj(c, V).*

4.3 Message refinement

Messages rejected in the above evaluation may still get accepted, provided some changes are made to them. Consider a strict policy that requires all messages to carry a monetary bond [3]. A sender may not know this requirement beforehand, and therefore, killing (*i.e.*, silently dropping) such a message, as shown earlier may cause dropping a desirable message. A better solution is to provide feedback to facilitate appropriate documentation to the message such that the evaluation policy can be satisfied. In our design such changes can be automatically enforced – by applying the sender's SP (send policy) and SESP's SPP (postman policy), where senders mandate their ESP's to some attributes to the message to satisfy the downstream policy. SESP can indicate this capability in the message by identifying headers that can be refined, *i.e.*, headers that cannot by changed are marked 'final', all others can be refined.

The clause sets $\mathcal{H}$ and $\mathcal{M}$ used when computing message refinements are generated as follows. for each header that is marked 'final', An *atrb* fact is created in the same manner used when determining whether a message is accepted. These facts constitute $\mathcal{H}$; no facts are created for non-final headers. We modify the original $\mathcal{M}$ by removing *syst* facts. This allows the value of environment variables to be set by the policy. With this change, policy evaluation can, for instance, inform the sender that the message is acceptable, but that transmission cannot be completed under current system conditions. Further, it can suggest when to retry transmission of an unaltered message. As before, facts in $\mathcal{M}$ are constructed using the *prim* predicates for each primitive mechanism that require final header values or the content of the message.

The clauses for the predicates rAllow and rDisallow are constructed from those for allow and disallow as follows. First, the head of each rAllow and rDisallow clause contains one variable for each header as well as two system variables. We add an equality between each of the header variables and the variable that appears in the body atom constructed with the *atrb* predicate for the same header. So given the vector of variables $\tilde{\mathrm{X}}$ in the head, we introduce $X_i = Y$ where $atrb_{Header_i}(Y)$ occurs in the original clause. Having done this, we then remove from the body any atoms constructed using $atrb_{Header_i}$ for each non-final header i. For the system variables, we add an equality between each of these variables and the variable that appears in the body atom constructed with the corresponding *syst* predicate.

Example 2. *Policy evaluation with message refinement*

Message rejected in previous example can be refined if marked as follows:

From: sender@abc.com (final); To: recipient@xyz.com (final); X-Auth: 'Password';

Here the SESP indicates that authentication can be upgraded, if required. atrb and prim facts are added to the policy as described earlier. We ignore syst predicates in the following. The rAllow predicate definition is:

$$rAllow(X_{\text{From}}, \ldots, X_{\text{Auth}}) \quad \leftarrow \quad X_{\text{Auth}} = \text{'PKI'}, atrb_{\text{From}}(sender), atrb_{\text{To}}(recipient).$$

The query succeeds and σ_a is given by X_{Auth} = 'PKI' $\wedge$ X_{from} = sender@ abc.com $\wedge$ X_{To}=recipient@xyz.com. This means that the message is acceptable if it is strongly authenticated. This result is made available to the SESP to refine the message, if desired.

■

4.3.1 Feasible refinements

Example 2 shows that if the SESP is incapable of providing a PKI based authentication of the sender, then it cannot document that the message is based on the feedback. Therefore, we allow a variation in the refining strategy to only provide feasible feedback. In this variation, the SESP is allowed to enumerate or provide a range values from which a solution must be selected. In example 2, the SESP can change the message as follows:

From: sender@abc.com (final); To: recipient@xyz.com (final); X-Auth: Password [Biometric, MAC-Address, IP-Address]

In this message, the SESP attaches a set of feasible values to the headers that can be refined further such as to Biometric authentication, *etc.* In general, *atrb* facts are constructed for all headers here, with 'final' headers each corresponding to a single fact in $\mathcal{H}$, and 'non-final' headers generating possibly multiple facts, or facts containing constraints rather than exact values. Again, the query $Q = \text{revise}(\tilde{\text{X}})$ is used, as in the previous case.

4.3.2 Selection of a suitable fix

The result of message refinement feedback generated by RESP is evaluated at the SESP to choose an optimum upgrade as it may involve monetary, computational or memory based costs. Next we present a simple distance based approach to select a minimum cost upgrade. We assume that the upgrade costs for equational constraints are available through a table lookup at the SESP. For interval constraints, if the new interval and current one intersect, the cost of upgrading is 0, otherwise, the cost assigned is the cost incurred to get an overlap between the two intervals. For example, to upgrade bond from [0,3] to [5,8], the cost is 5-3=2, but for upgrading [0,6], to [5,8] the cost is 0. We model upgrade costs through a function f that takes an original header and a revised header and returns a cost for upgrade. Original header values are available as a conjunction of unary constraints in the message, denoted by σ_b.

```
selectFix($\tilde{X}$, $\sigma_b$, $\sigma_a$, $f$):
Input: List of headers $\tilde{X}$
Input: Rejected message constraints $\sigma_b$
Input: Answer constraint, $\sigma_a$
Input: A cost function $f$ specified by the SESP
Output: A minimum cost fix to message
Let p = 0; $D_p = \infty$
For all disjuncts $d_j$ in $\sigma_a$
    dist = 0
    For each $X_i \in \tilde{X}$
        dist = dist + $f(proj(\sigma_b, X_i), proj(d_j, X_i))$
    If (dist < $D_p$) then
        p = j, $D_p$ = dist
return $d_p$
```

4.3.3 Policy control over feedback

Providing feedback for all rejected messages may be risky, as it can possibly lead to undesirable scenarios. The primary risk involves confirming the email identity of a recipient to a bulk-email business. Moreover, feedback may lead to undesirable messages getting through. We allow policies to suppress, or limit the amount of feedback provided. Amount of feedback provided depends upon the trustworthiness of senders or their SESPs. Based on the trust assessment, simple rules, modeled on MSP or MRAP syntax, can be used to control feedback. Prior history, distributed reputations, certificates, and other types of message documentation can be used to gauge the trustworthiness of principals.

4.4 Examples

In this section we present several example policies that show how to combine various email control mechanisms in our syntax. The policies we present are simple, non-recursive definitions for *allow* and *disallow* predicates, that use various other predicates. However, more complex policies, as deemed fit at a recipient domain, can be expressed in our language.

To combine different email control mechanisms, we use a predicate, suitably named, to make a 'call' to that mechanism. For example, *blacklist(X_{from})*, *whitelist(X_{from})* return *true* if X_{from} belongs to the respective list and $false$ otherwise. Similarly, *blocklist(X, URL)* models a call to a blocklist server like RBL [36], *spamNet(M)* calls SpamNet [22] for message *M*, *lumosRep(X), lumosPriority(M, P)* for Lumos attributes [34], *rep(X, URL, Val)* for general reputation services, *crm(M, Index)* [30] for calls to a spam filter. Example 3 presents some sample combinatorial MSP policies, where M denotes the complete message, and other predicates have usual meanings.

Example 3. *Combinations of available mechanisms*

1. *Lists [46, 36] with Monetary Bonds [3]*

$$\mathit{allow}() \leftarrow \mathit{atrb}_{\text{from}}(X), \mathit{whitelist}(X). \tag{4.1}$$

$$\mathit{allow}() \leftarrow \mathit{atrb}_{\text{from}}(X_1), \mathit{atrb}_{\text{Bond}}(X_2), X_2 \geq 2, \neg \mathit{blacklist}(X_1). \tag{4.2}$$

$$\mathit{allow}() \leftarrow \mathit{atrb}_{\text{Bond}}(X), X \geq 10. \tag{4.3}$$

$$\mathit{disallow}() \leftarrow \mathit{atrb}_{\text{from}}(X), \mathit{blocklist}(X, \text{`}\mathit{surbl.org}\text{'}). \tag{4.4}$$

2. Whitelist with CRM [30] filter

$$allow() \leftarrow prim_{\text{my-crm}}(I), I \leq 30. \tag{4.5}$$

$$allow() \leftarrow atrb_{\text{from}}(X), whitelist(X). \tag{4.6}$$

3. Authentication with reputations

$$allow() \leftarrow atrb_{\text{Auth}}(X), prim_{\text{RepService}}(V), V \geq 5, X = \text{‘}PKI'. \tag{4.7}$$

$$allow() \leftarrow atrb_{\text{Auth}}(X), prim_{\text{RepService}}(V), V \in [5,8], X = \text{‘}PKI'. \tag{4.8}$$

4. Virus Scanning with Lumos [34] reputations

$$disallow() \leftarrow prim_{\text{virusScan}}(\text{‘}Sobig.F'). \tag{4.9}$$

$$allow() \leftarrow prim_{\text{lumosRep}}(X), X \geq b. \tag{4.10}$$

5. Better service for partner domains

$$\begin{aligned} allow() &\leftarrow atrb_{\text{Auth}}(X_1), atrb_{\text{Sesp}}(X_2), partner(X_2), \\ &\quad X_1 = \text{‘}Password'. \end{aligned} \tag{4.11}$$

$$\begin{aligned} allow() &\leftarrow atrb_{\text{Auth}}(X_1), atrb_{\text{Sesp}}(X_2), partner(X_2), \\ &\quad X_1 = \text{‘}PKI'. \end{aligned} \tag{4.12}$$

$$\begin{aligned} disallow() &\leftarrow atrb_{\text{Sesp}}(X_1), syst_{\text{CurrTime}}(X_2), X_2 \in [9,12], \\ &\quad \neg partner(X_1). \end{aligned} \tag{4.13}$$

6. Allowing only high priority and sufficiently bonded mail

$$disallow() \leftarrow prim_{\text{lumosPriority}}(X), X \leq a_1. \tag{4.14}$$

$$disallow() \leftarrow prim_{\text{priorityServer}}(X), X \leq a_2. \tag{4.15}$$

$$disallow() \leftarrow atrb_{\text{Bond}}(X), X \leq a_3. \tag{4.16}$$

Policy combining whitelist/blacklist/blocklist with message bonds is shown in rules 4.1 through 4.4 in example 3 above. This policy accepts a message if the sender is in the whitelist (4.1) it will also accept the message if the message is bonded (for \$2.00) and the sender is not on recipient's blacklist (4.2) the policy also accepts a message irrespective of whether the sender is blacklisted or not, provided the message has sufficient bond value (\$10.00 in this case) while the policy disallows messages from any sender on the specified blocklist (4.4). Rules 4.5 and 4.6 show a policy that combines mail filter (CRM [30]) with a whitelist. This policy is quite helpful if a recipient does not wish to lose messages from whitelisted senders to the mail filter. Rule 4.5 essentially states the threshold for filtration. Rule 4.6 states that irrespective of the filter score, if the sender is a trusted person (whitelisted), then allow the message to go through. Rules 4.7 and 4.8 combine authentication criteria with sender reputations and filter scores. For instance, rule 4.7 states that if the sender has been strongly authenticated ('PKI' based authentication) and his/her reputation score is greater than 5 (say), then accept the message. Rule 4.8 amends rule 4.7 by giving a range of acceptable values for the reputation score. Rules 4.9 and 4.10 constitute a policy that combines the result of virus scanning with Lumos [34] reputations. Rule 4.9 states that a message should be rejected if the virus scan results state that the message contains ‘Sobig.F’ virus. Rule 4.10 states that only accept messages that have a Lumos reputation score of greater than a threshold (b). Rules 4.11 through 4.13 show how better service is made available to business partners. Rules 4.11 and 4.12 declare that partner SESPs can send messages from senders who are ‘password-authenticated’ or strongly authenticated (using PKI based authentication), *i.e.*, the choice is left to the sending domain. Rule 4.13 states that only partner domain’s mail servers can deliver mail between the hours of 9 am to 12 am. Finally, section 6 (rules 4.14 through 4.16) shows a stringent policy that allows very limited messages. Rule 4.14 states that all low priority messages (Lumos [34] priority less than a threshold, say a) should be rejected. Similarly, rule 4.15 states that low priority messages (priority fixed by a separate priority server) should be rejected; while rule 4.16 states that all messages with a bond value less than threshold (a_3) be rejected.

4.5 Evaluation and Complexity

In this section we show that the bottom up evaluation semantics is tractable. Additionally, the worst case time for computing σ_a and the worst case complexity for `selectFix` is PTIME.

In order to bound the complexity of calculating $T_P \uparrow \omega$ we will eliminate quantifiers in computed constraints, thereby limiting those constraints to ones that appear in the policy. Therefore, we restrict T_P to a smaller, but relevant subset, through an alternate definition of QE introduced below. This will induce the definition of T_P that is used in the complexity results below.

Li and Mitchell [47] show that $\mathcal{R}$ is *linearly decomposable*, *i.e.*, there exists a set C$'$ of basic constraints about a variable for any set of constraints C about the same variable, such that the conjunction of any set of constraints in C $\cup$ C$'$ can be represented by a disjunction of constraints in C$'$ and in which the size of C$'$ is linearly bounded by the size of C. Because $\mathcal{R}$ admits the negation symbol, C$'$ in our framework must be extended to include additional constraints to cover the entire constraint domain. Although the language allows a large set of constraints C, only a small subset is used in the evaluation. Hence, the size of the policy gives a stricter bound on the size of C$'$, *i.e.*, $|$C$'|\leq$d$|$P$|$, for a constant d.

Lemma 1. *Policy constraint domain admits existential quantifier elimination.*

Proof sketch: This follows from a similar theorem in [47]. Consider an arbitrary existential constraint. Any two constraints in the same variable, are either inconsistent, or the same (in C$'$). If inconsistent, the complete constraint is inconsistent, otherwise, we can reduce the original constraint by retaining only one of them. Hence, the given constraint can either be reduced to $false$ or a set of unary constraints, each in a different variable. We can simply drop the existentially quantified constraints. $\square$

Lemma 2. *Policy constraint domain admits universal quantifier elimination.*

Proof sketch: Consider an arbitrary constraint c in DNF form, universally quantified on some variables Y_k. Consider *proj*(c, Y_i)= c$_{Y_i}$. Each disjunct in c$_{Y_i}$ is either $false$ and can be removed, or reducible to a single unary constraint. If c$_{Y_i}$ is in $\mathcal{R}$, by sorting it's disjuncts on lower bounds of intervals and inspecting adjacent pairs, it can be verified that it "covers the entire domain" and hence is equivalent to $true$ thereby allowing us to drop $\forall Y_i$ from c; otherwise c is reduced to $false$. $\square$

The worst case time to evaluate the *proj* function is PTIME due to the absence of function symbols [44]. Clearly, quantifier elimination reduction has PTIME complexity. We redefine QE, such that it accepts any constraint in $\mathcal{R}$ as an argument, calculates its quantifier free DNF form in PTIME and returns the set of it's disjuncts. Thus, number of constraints calculated by $T_P \uparrow \omega$ can be bounded.

Theorem 1. *Assuming the number of variables in any literal in policy P is bounded by a constant, K, the worst-case complexity for computing σ_a is $|P|(d|P|)^K$, where d is a constant.*

Proof sketch: $T_P(I)$, and inductively $T_P \uparrow \omega$, only compute constrained atoms composed of atoms in the heads of clauses in the policy and constraints defined over their variables. The number of atoms, N_A, is bounded by the size of the policy, $N_A \leq |\mathrm{P}|$. If the size of the set of constraints is S, then N_S, the number of constraints, is less than or equal to S^K. S is given by $|C'| \leq \mathrm{d}|\mathrm{P}|$. Hence, $N_A \times N_S \leq |\mathrm{P}|(\mathrm{d}|\mathrm{P}|)^K$ $\square$

As described earlier, the maximum number of headers that can be refined, |X|, is negotiated between the SESP and RESP before the transmission of messages begin. Hence, K, the maximum number of variables in any literal is fixed. Therefore, even though K appears as an exponent in complexity analysis, the worst-case running time is still PTIME.

Theorem 2. *Worst case complexity for `selectFix` algorithm is PTIME.* $\square$

4.6 Chapter conclusion

In this chapter, we have proposed a CLP based policy syntax and semantics to express and enforce message acceptance preferences of the principals involved in email message transmissions. Our modeling enables a principal to effectively combine existing email control techniques to express diverse acceptance criteria, which can be precisely enforced without the risk of losing desirable messages, assuming that the feedback provided to a sender is utilized in refining rejected messages. This approach is a vast improvement over the current practice where desirable messages are discarded if they get flagged by spam filters. Though, in practice, an email message may use more than three hops to reach its destination, we consider three logical hops, corresponding to the control points. For the case when intermediate servers exist, several methods can be employed. For example, only designated servers evaluate policies, while other servers merely relay messages and policies in either direction, or, intermediate servers play the role of the missing party, *etc.*

In later chapters, we propose extensions to the SMTP protocol that enable our method to use most email control mechanisms, uniformly. Consequently, we are able to support a policy-controlled policy disclosures. This complements the basic idea of message refinement, by providing trusted senders an explicit criteria for message acceptance. This risks in leaking private information, such as, content of ESP maintained blacklists, whitelists *etc.*, but communicable policies can be sanitized before they are communicated, as shown later. With out of band policy disclosures, concerns over bandwidth required to transmit large policies can be addressed.

Chapter 5: A Formal Model for Generalized Messages

5.1 Introduction

In this chapter we generalize the formal model introduced in the last chapter. The aim here is to present an end to end logical model of message transmission, *i.e.*, we extend policy syntax to allow specification of all six policies introduced in chapter 3; and show how policy evaluations can aid/prevent the delivery of desirable/undesirable (resp.) messages. In other words, actions like selectFix, refine, reject, accept, *etc.*, discussed previously will be formalized into a single logical framework. Intuitively, the system that we describe here can be considered as a collection of constraint logic programs that are to be run at appropriate agents. These clusters, called *modules*, interact with each other by exporting and importing predicates (cf. Maher [48]), just like traditional distributed systems export interfaces for invocation of imperative procedures. The collection of these modules is called a *module network*, formalized later. The overall logical framework is represented in figure 5.1).

Figure 5.1 pictures a module network comprising of four policies: SP, SPP(= SPP-DM ∪ SPP-RM), MSP and MRAP. Each module shown interacts with other modules by importing or exporting predicates. Each directed arrow identifies the source and destination of the predicate that is identified by the arrow label(s). These interactions are designed to provide the extended SMTP functionality, *i.e.*, support for expressing

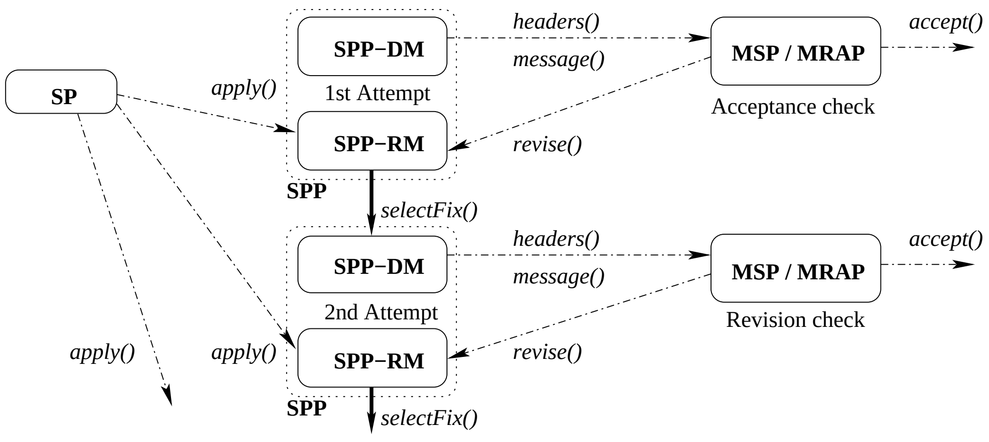

Figure 5.1: Interaction of policies for message transmission

email-control preferences, support for temporarily rejecting a message, support for refining a message, and support for selecting appropriate option among the set of ways to fix a temporarily rejected message. We don't support policy advertisement here for obvious reasons. The top-level query is 'accept()', which forms the interface to the module network. A message is represented as a set of facts in the SPP-DM, SPP's delivery module. SPP-DM exports 'message()' and 'header()' predicates to MRAP and MSP policies, respectively. MSP and MRAP can 'accept()' a message or reject it by exporting 'revise()' predicate to SPP-RM, the revision module. SP's 'canChange()' predicate indicates what changes can be applied. Using the 'canChange()' predicate, SPP-RM computes an inexpensive revision, by using the 'selectFix()' predicate. The revised message is then enqueued for retransmission. The '1st Attempt' and '2nd Attempt' labels denote initial transmission and retransmission of a revised message to emphasize the fact that the predicate communication graph is acyclic. In essence, a re-attempt can be treated as a new message.

5.2 Formal Model

5.2.1 Syntax

Logical elements like terms, constraints, *etc.*, are usually defined (please refer to chapter 4 definitions 1 –10); here we extend those definitions to generalize the formal model.

Definition 17 (Predicates). *Predicate symbols (described in tables 5.1, 5.2, and 5.3) are partitioned into following sets:*

- *Required local predicates (RL) (See table 5.1)*
- *Import-Export predicates (IE) (See table 5.2)*
- *Optional local predicates (OL) (See table 5.3)*

The intuitive meanings of predicates introduced in definition 17 are presented in tables 5.1,5.2,5.3. Readers may wish to peruse the table as they read the definition above.

Definition 18 (Clause, Fact, Rule). *A clause is of the form $H \leftarrow B$ where H is an atom, and B is a list of literals. A clause is called a fact if $B = \lambda$ (empty list), and a rule otherwise.*

Definition 19 (Policy Module). *A policy module (or simply a module), M_P, is a set of clauses P_P with three disjoint sets of predicates: the local predicates, Loc_P, the exported predicates, Exp_P, and the imported predicates Imp_P. We require that the head of clauses in P_P be from $Loc_P \cup Exp_P$.*

Table 5.1: Required Local Predicates (RL)

Predicate/ Arity	**Description**
allow/6	defines acceptance criteria; arguments correspond to header values and delivery iteration
disallow/6	defines rejection criteria, same arguments as allow
rAllow/6	acceptable revised header tuples
rDisallow/6	unacceptable revised header tuples
Env/2	sets non-revisable header variable to the value given in the message
checkEnv/6	collects header values in the original message
fixEnv/6	collects non-revisable header values
icost/4, cost/4	cost for revising a header H with having value R_1 to new value R_2
total/13	total cost for revising headers $\overrightarrow{R_1}$ to $\overrightarrow{R_2}$
selectFix/6	computes a low-cost revision to a message
relay/7	defines criteria for relaying a message; arguments correspond to header values, content and delivery iteration
norelay/7	defines criteria for dropping a message; same arguments as relay
deliver/7	combines relay and norelay
canChange/2	permits change to header value
nHeader/2	associates header names with values

Next we define the policy modules that we construct using the above syntax. In the following, we abuse the terminology at times and call an atom a predicate (to be able to discuss predicates and their arguments).

Definition 20 (MSP policy module)**.** *An MSP policy module M_{MSP}, is given by $\langle Imp_{MSP}$, Exp_{MSP},Loc_{MSP},$P_{MSP}\rangle$, in which Imp_{MSP}= {header, nonFinal, sys_{Var_i}, $prim_{Mech_j}$}, Exp_{MSP}={ accept, revise}, Loc_{MSP}={allow, disallow, rAllow, rDisallow, checkEnv, fixEnv, Env} $\cup$ OL predicates and P_{MSP}, a set of facts and stratified [49] rules, which has four strata:*

Table 5.2: Import-Export Predicates (IE)

Predicate/ Arity	**Description**
accept/6	combines allow & disallow, with same arguments as allow
revise/12	combines rAllow & rDisallow
header/2	associates header names with original values
content/1	associates 'content' with message provided value
message/7	associates header names and 'content' with values
nonFinal/1	True if header can be revised
$\mathrm{sys}_{\mathrm{var}}/2$	associates system variable 'var' to current value
$\mathrm{prim}_{\mathrm{mech}}/2$	associates computation of 'mech' to the computed value

Table 5.3: Optional Local Predicates (OL) (Examples)

Predicate/ Arity	**Description**
whitelist/1	True if argument is whitelisted
blacklist/1	True if argument is blacklisted
partner/1	True if argument is a partner

- **Stratum 0:** *Definitions of RL predicates: checkEnv($\overrightarrow{X}$), fixEnv($\overrightarrow{X}$) and Env(X,Y), where* $\overrightarrow{X} = X_{From}, \ldots, X_{Auth}$. *These are as follows:*

$$checkEnv(\overrightarrow{X}) \leftarrow header(From, X_{From}), header(To, X_{To}), header(Date, X_{Date}), header(Sesp, X_{Sesp}), header(Bond, X_{Bond}), header(Auth, X_{Auth}). \tag{5.1}$$

$$fixEnv(\overrightarrow{X}) \leftarrow Env(From, X_{From}), \ldots, Env(Auth, X_{Auth}) \tag{5.2}$$

$$Env(H, X_H) \leftarrow nonfinal(H) \tag{5.3}$$

$$Env(H, X_H) \leftarrow header(H, X_H) \tag{5.4}$$

- **Stratum 1:** *Definitions of OL predicates*

- **Stratum 2:** *Definition of following RL predicates: allow($\overrightarrow{X}$), disallow($\overrightarrow{X}$), rAllow($\overrightarrow{X}$) and rDisallow($\overrightarrow{X}$). In addition to Strata 0,1 predicates, most of IE predicates can be used in the body of the defined rules, except content(X), message($\overrightarrow{X}$), and prim*$_{\text{mech}}$*(X, Y) predicates. As a further restriction,* allow *and* disallow *predicates cannot use* fix-Env *or* Env *predicates.* rAllow *and* rDisallow *are constructed from* allow *and* disallow *predicate definitions in the following manner. For each allow (resp., disallow) rule, a rule with rAllow head (resp., rDisallow) is generated with checkEnv replaced by fixEnv.*

- **Stratum 3:** *RL predicate accept, defined as, accept*($\overrightarrow{X}$) $\leftarrow$ *allow*($\overrightarrow{X}$), $\neg$*disallow*($\overrightarrow{X}$)*;*

- **Stratum 4:** *RL predicate revise, defined as, revise($\overrightarrow{X}$,$\overrightarrow{Y}$) $\leftarrow$ checkEnv($\overrightarrow{X}$), $\neg$accept($\overrightarrow{X}$), rAllow($\overrightarrow{Y}$), $\neg$ rDisallow*($\overrightarrow{Y}$)*.*

Note that constraints do not occur in the rules (5.1– 5.4) as Env, checkEnv, and fixEnv predicates are used to set up the environment, *i.e.*, the values of headers provided in the message. Constraints on variables are expressed in the rest of the policy. Negation is used in a limited way, such as, in the definitions of system defined predicates `accept, deliver`, or with `allow` and `disallow`, *etc.* Negation can be used in OL predicates, provided some conditions are met. Primarily, OL predicate definitions must be stratified according to the following restrictions (i) can positively refer to atoms whose predicate symbols are defined in the same or lower OL strata, and atoms in Stratum 0 (ii) can negatively refer to atoms whose predicate symbols are defined in a lower OL strata and atoms in Stratum 0. The OL strata forms a substrata of Stratum 1.

Definition 21 (MRAP policy module)**.** *An MRAP policy module, M_{MRAP} is given by*

$\langle$*Imp*$_{MRAP}$, *Exp*$_{MRAP}$,*Loc*$_{MRAP}$,$P_{MRAP}\rangle$, *where*

1. *Imp*$_{MRAP}$=$\{$ *content, message*$\}$ $\cup$ *Imp*$_{MSP}$,
2. *Exp*$_{MRAP}$= $\{$*accept*$_a$, *revise*$_a\}$,
3. *Loc*$_{MRAP}$ = $\{$*allow*$_a$, *disallow*$_a$, *rAllow*$_a$, *rDisallow*$_a$, *checkEnv*$_a$,*fixEnv*$_a$,*Env*$_a\}$ *and*
4. P_{MRAP} *is a set of stratified clauses, with the following strata (all Loc and Exp predicates have been subscripted to prevent name collisions with MSP policy):*

- **Stratum 0:** *Definitions of RL predicates checkEnv*$_a(\overrightarrow{X})$, *fixEnv*$_a(\overrightarrow{X})$ *and Env*$_a(X,Y)$, *as defined in the MSP stratum 0.*
- **Stratum 1:** *Definitions of OL predicates.*
- **Stratum 2:** *Definitions of the following RL predicates: allow*$_a(\overrightarrow{X})$, *disallow*$_a(\overrightarrow{X})$, *rAllow*$_a(\overrightarrow{X})$ *and rDisallow*$_a(\overrightarrow{X})$. *All IE predicates can be used in the body of the defined rules. MSP restrictions on use of* fixEnv$_a$ *apply.* rAllow$_a$, rDisallow$_a$ *are constructed as shown in MSP stratum 2.*
- **Strata 3 and 4:** *RL predicate accept*$_a(\overrightarrow{X})$ *(Stratum 3) and revise*$_a(\overrightarrow{X},\overrightarrow{Y})$ *(Stratum 4), defined similar to the corresponding predicate definitions in MSP strata 3 and 4.*

Definition 22 (SP policy module)**.** *SP policy module,* $M_{SP} = \langle$*Imp*$_{SP}$, *Exp*$_{SP}$,*Loc*$_{SP}$,$P_{SP}\rangle$, *in which Imp*$_{SP}$ =$\emptyset$, *Exp*$_{SP}$=$\{$ *canChange*$\}$, *Loc*$_{SP}$ =$\emptyset$ *and* P_{SP}, *is a set of rules or facts that define canChange*(X,Y) *clauses.*

Definition 23 (SPP)**.** *SPP consists of two modules:*

SPP-DM: Delivery module *Delivery module, $M_{SPP\text{-}DM} = \langle Imp_{SPP\text{-}DM}, Exp_{SPP\text{-}DM}, Loc_{SPP\text{-}DM}, P_{SPP\text{-}DM}\rangle$, in which $Imp_{SPP\text{-}DM}$=$\{canChange\}$, $Exp_{SPP\text{-}DM}$=$\{header, content, message, nonFinal\}$, $Loc_{SPP\text{-}DM}$=$\{relay, norelay, deliver\}$ and $P_{SPP\text{-}DM}$, consists of following strata of rules:*

- **Stratum 0:** *Facts - header(X,Y), content(X) and nonFinal(X) and definition of message($\overrightarrow{X}$, C) predicate: message($\overrightarrow{X}$, C) $\leftarrow$ header(From, X_{From}), ..., header(Auth, X_{Auth}), content(C); where $\overrightarrow{X} = X_{From}, \ldots, X_{Auth}$.*
- **Stratum 1:** *Definition of RL predicates relay($\overrightarrow{X}$, C) and norelay($\overrightarrow{X}$, C). Definitions can use IE predicates and lower strata predicates in the body.*
- **Stratum 2:** *Definition of SP predicate deliver($\overrightarrow{X}$, C), defined as, $deliver(\overrightarrow{X},C) \leftarrow relay(\overrightarrow{X},C), \neg norelay(\overrightarrow{X},C)$*

SPP-RM Revision module *Revision module, $M_{SPP\text{-}RM}$=$\langle Imp_{SPP\text{-}RM}, Exp_{SPP\text{-}RM}, Loc_{SPP\text{-}RM}, P_{SPP\text{-}R}$ in which $Imp_{SPP\text{-}RM}$=$\{revise, revise_a\}$, $Exp_{SPP\text{-}RM}$= $\emptyset$, $Loc_{SPP\text{-}RM}$= $\{cost, icost, total, selectFix\}$ and $P_{SPP\text{-}RM}$, a set of rules that define selectFix($\overrightarrow{X}$), total($\overrightarrow{X}$, $\overrightarrow{Y}$, Z), cost(H,X,Y,C) and icost(H,X,Y,C) predicates. The icost(H,X,Y,C) predicate is defined by a collection of facts that define a cost C to change header value X to Y. The predicates nHeader and cost are defined as (maxInt is the largest integer in $\mathcal{FD}$):*

$$cost(H,X,Y,C) \leftarrow canChange(H,Y), icost(H,X,Y,C) \tag{5.5}$$

$$cost(H,X,Y,maxInt) \leftarrow \neg canChange(H,Y) \tag{5.6}$$

$$nHeader(H,X_H) \leftarrow selectFix(X_{From},\ldots,X_H,\ldots,X_{Auth}) \tag{5.7}$$

$$nHeader(H,X_H) \leftarrow \neg selectFix(X_{From},\ldots,X_H,\ldots,X_{Auth}), nHeader(H,X_H) \tag{5.8}$$

Definition 24 (Supporting modules). *Supporting policy modules include the system module, a four tuple:* $\langle \emptyset, \{sys_{var_i}\}, \emptyset, F_s \rangle$*, and primitive mechanisms, which are given by following forms of 4 tuples:* $\langle \{header(h, X_h)\}, \{prim_{mech_i}\}, \emptyset, F_p \rangle$*.*

The sets of clauses (F_s, *etc.*) in the supporting modules are essentially a set of facts. In practice, these are constructed at evaluation time based on system environment conditions. The number of supporting modules varies depending on the predicates used in the MSP and the MRAP modules. We use a set SM$_n$, to represent *all* the supporting modules whose predicates are imported by the MSP and the MRAP modules.

Example 4 (MSP or MRAP policy module). *Consider the Stratum 3:*

$$allow(\overrightarrow{X}) \leftarrow checkEnv(\overrightarrow{X}), whitelist(X_{From}). \tag{5.9}$$

$$disallow(\overrightarrow{X}) \leftarrow checkEnv(\overrightarrow{X}), blacklist(X_{From}). \tag{5.10}$$

$$allow(\overrightarrow{X}) \leftarrow checkEnv(\overrightarrow{X}), X_{Auth} = PKI. \tag{5.11}$$

Rule 9 says that a message is acceptable if the sender is in the recipient's whitelist. Rule 10 says that a message is unacceptable if the sender is found to be on the recipient's blacklist. Rule 11 says that a message is acceptable whenever the message has been strongly authenticated. Together, the rules allow emails from senders who are whitelisted or who strongly authenticate their messages, and block messages from blacklisted senders.

Example 5 (SP policy module).

$$canChange(Bond, C) \leftarrow C < 5 \tag{5.12}$$

$$canChange(Auth, `PKI') \leftarrow \tag{5.13}$$

Rule 12 says that if required, a bond of value less than 5 can be applied to the outgoing message. Similarly, fact 13 says that sender's private key can be used to add digital signatures or other forms of PKI based authentication.

Example 6 (SPP). *A simple delivery module (SPP-DM):*

$$relay(\overrightarrow{X},C) \leftarrow message(\overrightarrow{X},C), \neg prim_{Nortan}(C) \tag{5.14}$$

$$relay(\overrightarrow{X},C) \leftarrow message(\overrightarrow{X},C), \neg prim_{crm2}(C,X), X < 0.3 \tag{5.15}$$

Rule 14 says that if the message is found free of any virus, it can be delivered. Rule 15 states a message ranking low on the spam filter, can be delivered. Together the rules require a message to undergo either a virus-scan or a filtering process to be allowed delivery.
A simple revision module (RM):

$$\begin{aligned} total(\overrightarrow{R_1},\overrightarrow{R_2},\overrightarrow{C}) &\leftarrow revise(\overrightarrow{R_1},\overrightarrow{R_2}), cost(From, R_{1,From}, R_{2,From}, C_1), \ldots, \\ &\quad cost(Auth, R_{1,Auth}, R_{2,Auth}, C_6) \end{aligned} \tag{5.16}$$

$$selectFix(\overrightarrow{R_2}) \leftarrow minimize(total(\overrightarrow{R_1},\overrightarrow{R_2},\overrightarrow{C}), C_1 + \ldots + C_6). \tag{5.17}$$

Rule 16 shows how to calculate the costs to revise a message using the cost predicate and vectors $\overrightarrow{R_1},\overrightarrow{R_2}$ whose individual elements are $R_{1,From}$, $R_{2,From}$, etc. and C_j is the cost for changing $R_{1,j}$ to $R_{2,j}$. Also, $\overrightarrow{C} = C_1, \ldots, C_6$. Rule 17 calculates the minimum cost change by minimizing the total cost of revision.

Definition 25 (System of policy modules). *A system of policy modules, Γ, is given by a finite set of policy modules, indexed by I. Thus M_i ranges over modules in Γ for $i \in I$. For any $i,j \in I$, $i \neq j$, we have $Loc_i \cap Loc_j = \emptyset \wedge Exp_i \cap Exp_j = \emptyset$. For such i,j we write $i \sqsubset_\Gamma j$ if $Exp_i \cap Imp_j \neq \emptyset$. Letting $\sqsubset_\Gamma^*$ denote the transitive closure of $\sqsubset_\Gamma$, we require that $\forall$*

$i \in I$, $i \not\sqsubset^*_\Gamma i$ (irreflexivity), i.e., the relation $\sqsubset^*_\Gamma$ is a partial order.

Definition 26 (Complete system of policy modules)**.** *A system of policy modules is a complete system of policy modules if $\forall\ i\ \in\ I\ Imp_i \subset \bigcup_{j\in I\wedge j\neq i} Exp_j$*

Theorem 1. *A set of policy modules consisting of one of each policy module kinds: SP, SPP-DM, SPP-RM, MSP, MRAP and SM_n forms a complete system of policy modules.*

Proof: We give an informal proof to show that the set of policy modules $\Gamma_1 = \{\mathrm{M_{SP}}, \mathrm{M_{SPP\text{-}DM}}, \mathrm{M_{SPP\text{-}RM}}, \mathrm{M_{MSP}}, \mathrm{M_{MRAP}}\} \cup \mathrm{SM}_n$ satisfies all the criteria to be a complete system of policy modules. Following properties can be ascertained in a straightforward manner.

1. Firstly, for each pair of distinct policy modules M_i and M_j $(i, j$ range over an index I_1 of $\Gamma_1)$, $\mathrm{Loc}_i \cap \mathrm{Loc}_j = \emptyset \wedge \mathrm{Exp}_i \cap \mathrm{Exp}_j = \emptyset$ holds.

2. Secondly, the relation $\sqsubset_{\Gamma_1}$ is irreflexive and transitive (*i.e.*, the condition $\forall\ i \in \mathrm{I}_1$, $\nexists\ \mathrm{k}_1,\ldots,\mathrm{k}_n \in \mathrm{I}_1$ such that $i \sqsubset^*_{\Gamma_1} i$, can be easily verified).

3. Finally, to see the completeness property, we construct a set $\bigcup_{j\in I_1} Imp_j$ and verify that $\bigcup_{j\in I_1} Imp_j \subset \bigcup_{k\in I_1} Exp_k$. □

Notation 1 (System of email policy modules)**.** *A complete system of email policy modules is represented by Γ_{em}, which is given by the set $\{M_i, SM_j \mid i \in I_1$ and $j \in I_2\}$, where $I_1 = \{$SP, SPP-DM, SPP-RM, MSP, MRAP$\}$ and $I_2 = \{1, \ldots, k\}$, $SM_1, \ldots, SM_k \in SM_n$ and each M_i, SM_j satisfies the appropriate definitions, 9 – 13, above. The combination of these indices is represented by $I_{em} = I_1 \cup I_2$.*

Theorem 2 (Stratification [49]). *$\forall\ i\ \in I$, the sets P_i and $\bigcup_i\ P_i$ are stratified.*

Proof: Stratification for MSP may be obtained through the following level mapping [49, 50] ℓ.

1. ℓ assigns imported predicate headers, nonFinal, prim_{Mech_j} and syst_{Var_k}, to the level 1.

2. Predicates checkEnv, fixEnv and Env are assigned the level 2;

3. 3 is assigned to all OL predicates;

4. 4 is assigned to the RL predicates defined in Stratum 2 of the policy;

5. 5 is assigned to accept and 6 to revise.

MRAP Stratification is obtained in a similar manner as above, with the addition of message, content, prim_{Nortan} and prim_{crm1} to level 1.

Following level mapping, ℓ, assigns strata to SPP predicates as follows: 0 is assigned to header, content, nonFinal, and icost. Level 1 is assigned cost. Level 6 is assigned to the IE predicates revise, revise_a and RL predicates total, selectFix; nHeader is assigned the level 7; relay, norelay to 8 and deliver is assigned to level 9.

The level mapping for SP is trivial, with canChange being assigned the level 0. Similarly, all predicates defined in SM_n modules are assigned to 0. The level mapping for $\bigcup_i$ P_i is simply the union of the level mappings for individual policies. $\square$

5.2.2 Semantics

We use three-valued Kunen-Fitting (or Kripke-Kleene) [51] semantics, for interpreting our CLP programs. We use constructive negation [52] as proposed by Fages [45]. We first repeat some standard definitions as they appear in [51] and are repeated in [53].

Definition 27 (P*, T$_P$ and $\Phi_P \uparrow$ operators)**.** *Suppose P is a policy module, and let P^* be all ground instances of clauses in P. We now define two and three valued truth lattices to be* $2 = \langle\{T,F\}, <_2\rangle$ *and* $3 = \langle\{T,F,\perp\} <_3\rangle$ *respectively, where T, F and $\perp$ are taken to mean true, false and unknown truth values. Partial orderings $<_2$ and $<_3$ satisfy $F <_2 T$ and $\perp <_3 T$, $\perp <_3 F$ respectively. A mapping V from the herbrand base of P to* 2 *or (respectively* 3*) is said to be a two-valued (respectively a three-valued) valuation of P. Any valuation is naturally extended to negative literals according to the following interpretation of negation: $\neg T = F$, $\neg F = T$ and $\neg\perp = \perp$. Also, $\alpha \vee \beta = T$ if $\alpha = T$ or $\beta = T$; $\alpha \vee \beta = F$ if $\alpha = F$ and $\beta = F$; and $\alpha \vee \beta = \perp$ otherwise. $\vee$ extends pointwise to valuations. Given a valuations V_l and V_i, the two and three valued immediate consequence operators $T_P^{V_i}(V_l,\ V_i)$ and $\Phi_P^{V_i}(V_l,\ V_i)$ are defined as follows:*

$T_P^{V_i}(V_l, V_i)$: $T_P^{V_i}(V_l, V_i) = W$ is defined as

- *$W(H) = T$ if there is a ground clause $H \leftarrow B$ in P^* such that $V_i(B_k) = T$ for all $B_k \in B$ constructed using a predicate in Imp_P and $V_l(B_m) = T$ for all $B_m \in B$ constructed using predicates not in Imp_P.*
- *$W(H) = F$ otherwise.*

$\Phi_P^{V_i}(V_l, V_i)$: $\Phi_P^{V_i}(V_l, V_i) = W$ is defined as

- *$W(H) = T$ if there is a ground clause $H \leftarrow B$ in P^* such that $V_i(B_k) = T$ for all $B_k \in B$ constructed using a predicate in Imp_P and $V_l(B_m) = T$ for all $B_m \in B$ constructed using predicates not in Imp_P.*
- *$W(H) = F$ if for every ground clause $H \leftarrow B$ in P^*, $V_i(B_k) = F$ for some $B_k \in B$ constructed using a predicate in Imp_P or $V_l(B_m) = F$ for some $B_m \in B$ constructed*

using predicates not in Imp_P.

- *$W(H) = \perp$ otherwise.*

Now we define bottom-up semantics for both T_P and Φ_P, where Ψ stands for either of them in the following:

- $\Psi_P^{V_i} \uparrow (0) = V_{false}$*, where V_{false} assigns F (false) to all instantiated atoms.*
- $\Psi_P^{V_i} \uparrow (\alpha + 1) = \Psi_P^{V_i}(\Psi_P^{V_i} \uparrow (\alpha), V_i)$ *for every successor ordinal α.*
- $\Psi_P^{V_i} \uparrow (\alpha) = \bigvee_{\beta<\alpha} (\Psi_P^{V_i} \uparrow (\beta))$ *for limit ordinal α.*

Definition 28 (Bottom-up semantics)**.** *Let $P_i \in \Gamma$ be a policy module and Φ be the three valued consequence operator as defined above. We let $\Phi_{P_i}^{V_i} \uparrow (\omega)$ be the semantics of P.*

Definition 29 (Projection operator)**.** *Given a valuation V and a set of predicates P such that V is defined over atoms constructed using predicates in P and possibly some other predicates, projection $V|_P$ is the valuation defined over only atoms constructed using predicates in P and having the same value as V on those atoms.*

An example MSP Evaluation

Next we present a simple MSP policy module evaluation in an example.

Example 7. *Policy module evaluation without feedback*

Consider a message with only three headers:

Mail From: sender@abc.com (final); Rcpt To: recipient@xyz.com (final); X-Auth: 'Password'(final);

The Keyword 'final' in the headers indicates that they cannot be revised. We show the evaluation of the module described in example 4 next. We assume that the sender does not belong to either the whitelist or the blacklist. In the $\Phi_{P_{\text{MSP}}}^{V_i}$ computation, presented in table 5.4, we omit the '@abc.com' part in sender address, and predicates like *rAllow*, *rDisallow*, *revise* and *fixEnv*. We use meta-variable $\overrightarrow{x}$ to refer to a tuple of constants, *i.e.*, header values. Also, we only show the whitelist/blacklist predicate instances for sender@abc.com.

Table 5.4: $\Phi_{P_{\text{MSP}}}^{V_i}$ calculation for message acceptance

Ord	$W(H)$=**T**	$W(H)$=**F**
1	{cHeader(Auth,'Password'), cHeader(From, 'sender'), …}	{whitelist('sender'), blacklist('sender'),…, nonFinal(Auth), nonFinal(From),nonFinal(To)}
2	{cHeader(Auth,'Password'), cHeader(From, 'sender'), …, checkEnv($\overrightarrow{x}$) }	{whitelist('sender'), blacklist('sender'),…, nonFinal(Auth), …}
3	{cHeader(Auth,'Password'), cHeader(From, 'sender'), …, checkEnv($\overrightarrow{x}$) }	{allow($\overrightarrow{x}$), disallow($\overrightarrow{x}$), whitelist('sender'), blacklist('sender'),…, nonFinal(Auth), …}
4	{cHeader(Auth,'Password'), cHeader(From, 'sender'), …, checkEnv($\overrightarrow{x}$) }	{ accept($\overrightarrow{x}$), allow($\overrightarrow{x}$), disallow($\overrightarrow{x}$), whitelist('sender'), blacklist('sender'),…, nonFinal(Auth), …}

The fixpoint is reached at ordinal 4. Since $\Phi_{P_{\text{MSP}}}^{V_i} \uparrow (4)$ cannot prove *accept*, the message is rejected. The root cause is that the authentication provided in the message is inconsistent with that required in the policy.

Message revision

The messages rejected, as in example 7, may be desirable to the recipient, and should not be dropped without letting the sender know why they were dropped. If this information is provided to the SPP, it can appropriately revise an initially rejected message and, thus, successfully deliver it. To do so, we require a minor change in how messages are documented. Message headers are required to include a qualifier, namely, 'final', for every header that cannot be revised. This indicates to the MSP or the MRAP which headers can be revised. This documentation is captured by the nonFinal(H) atom. Using the fixEnv($\overrightarrow{X}$) atom in the definitions of constructed predicates rAllow($\overrightarrow{X}$) and rDisAllow($\overrightarrow{X}$), policy-desired constraints can be captured in *revise* predicate. Ground instances of *revise* are exported by the evaluating module to indicate to the SPP, the desired documentation in a rejected message.

Example 8. *Policy module evaluation with message revision*

Message rejected in example 7 can be revised if marked as follows:

Mail From: sender@abc.com (final); Rcpt To: recipient@xyz.com (final); X-Auth: 'Password';

Here the SESP indicates that authentication can be upgraded, if required. Next we show the evaluation of the policy rules to revise this rejected message. The $\Phi^{V_i}_{P_{\text{MSP}}}$ computation is presented in table 5.5. The fixpoint is reached at ordinal 5 and $\Phi^{V_i}_{P_{\text{MSP}}} \uparrow (5)$ proves a revise($\overrightarrow{X}$, $\overrightarrow{Y}$) atom.

Table 5.5: $\Phi^{V_i}_{P_{\text{MSP}}}$ calculation for message revision

Ord	$W(H)$=**T**	$W(H)$=**F**
1	{nonFinal(Auth), cHeader(From,'sender'), cHeader(To,'recipient'), cHeader(Auth,'Password')}	{nonFinal(From), nonFinal(To), whitelist(sender), blacklist(sender), ... }
2	{nonFinal(Auth), cHeader(From,'sender'), ..., checkEnv($\overrightarrow{x}$), Env(From,'sender'), Env(To,'recipient'), Env(Auth,_)}	{nonFinal(From), nonFinal(To), whitelist(sender), blacklist(sender), ... }
3	{nonFinal(Auth), cHeader(From,'sender'), ..., checkEnv($\overrightarrow{x}$), Env(From,'sender'), ..., fixEnv($\overrightarrow{x}$)}	{allow($\overrightarrow{x}$), disallow($\overrightarrow{x}$), nonFinal(From), nonFinal(To), whitelist(sender), blacklist(sender), ... }
4	{nonFinal(Auth), cHeader(From,'sender'), ..., checkEnv($\overrightarrow{x}$), Env(From,'sender'), ..., fixEnv($\overrightarrow{x}$), *rAllow* ($\overrightarrow{x}$)}	{allow($\overrightarrow{x}$), disallow($\overrightarrow{x}$), nonFinal(From), nonFinal(To), whitelist(sender), blacklist(sender), ..., accept($\overrightarrow{x}$), *rDisallow* ($\overrightarrow{x}$)}
5	{nonFinal(Auth), cHeader(From,'sender'), ..., checkEnv($\overrightarrow{x}$), Env(From,'sender'), ..., fixEnv($\overrightarrow{x}$), *rAllow* ($\overrightarrow{x}$), *revise* ($\overrightarrow{x}$,$\overrightarrow{y}$)}	{allow($\overrightarrow{x}$), disallow($\overrightarrow{x}$), nonFinal(From), nonFinal(To), whitelist(sender), blacklist(sender), ..., accept($\overrightarrow{x}$), *rDisallow* ($\overrightarrow{x}$)}

5.3 Materialization Structure

Section 5.2 establishes stratification [49] of each local policy module and their union. This essentially allows each delivery attempt being treated as a new message delivery. Thus, for a single message delivery attempt, the module graph is acyclic, as required in [48]. There can be many strategies for exporting predicates, like, for instance, 'one-at-a-time' transmission or 'all-together' transmission strategies. In our view, the second approach can be more efficient in reducing communication overheads, and hence we propose materializing exported predicates to support this strategy.

Definition 30 (Materialization Structure $\mathcal{MS}$)**.** *The materialization structure, $\mathcal{MS}_i$, of a module M_i is a valuation over atoms constructed using the Exp_i predicates.*

Definition 31 (Correctness)**.** *Given a system of policy modules* Γ *indexed by I, a corresponding set of materialization structures also indexed by I, and ranged over by* $\mathcal{MS}_i$*, is correct if* $\forall\ i \in I\ \mathcal{MS}_i = \Phi_{P_i}^{V_i} \uparrow (\omega) \mid_{Exp_i}$*, where* $V_i = \bigcup_{j \sqsubset_\Gamma i} (\mathcal{MS}_j \mid_{Imp_i})$ *(Here we are viewing the function* $\mathcal{MS}_j$ *as sets of pairs. In this way combine these functions whose domains are disjoint).*

Theorem 3 (Faithfulness and Adequacy)**.** *Given a complete system of policy modules* Γ*, indexed by I, and a corresponding set of materialization structures also indexed by I and ranged over by* $\mathcal{MS}_i$*,*$\bigcup_{i \in I} \mathcal{MS}_i = \Phi_P^{\emptyset} \uparrow (\omega) \mid_{Exp}$*, where* $P = \bigcup_{j \in I} P_j$*,* $Exp = \bigcup_{j \in I} Exp_j$*.*

Proof Strategy: The proof strategy of using induction over i would give the required proof. □

Due to theorem 1, theorem 3 applies to the complete system of email policies, which is significant for our application. The communication of imported-exported predicates between modules is proposed through a materialization of individual (local) modules. The evaluation of the next module, *i.e.*, the module whose predecessor(s) has (have) completed its (their) evaluation(s), depends upon the inputs from predecessors. The significance of theorem 3 lies in the fact that it shows the correctness of the 'relative' (local) evaluation scheme with respect to an evaluation scheme with the possession of global knowledge.

5.4 Chapter conclusion

In this chapter we generalize the formal model for email policies to provide an end to end formal representation of email messages and the participation of email delivery agents in message transmission. Policies are presented as *module networks*, *i.e.*, a network of programs that can import or export *predicates*, (*c.f.* Maher [48]). Using predicate import-export email delivery agents can transmit messages and feedback regarding rejected messages.

Chapter 6: Privacy

6.1 Introduction

This chapter identifies undesirable side-effects of combining different email-control mechanisms for protection from unwanted messages, namely, leakage of recipients' private information to message senders. The problem arises because some email-control mechanisms like bonds [3], graph-turing tests [4], *etc.*, inherently leak information. We show how an attacker can guess recipient's mail acceptance policy that utilizes leaky mechanism in an effort to avoid unwanted mail.

Leakages can occur in many ways. For example, simple *address harvesting* attacks can be constructed through the Simple Mail Transfer Protocol (SMTP [1]), easily. In this attack, a malicious sender attempts delivery to a preconstructed list of possible recipient addresses, and recipient mail server replies help her to identify which addresses are assigned to users [54]. Contrary to the SMTP protocol recommendations, mail servers can prohibit such feedback, thus implementing a blanket protection policy against harvesting attacks. More fine-tuned, policy-based schemes for feedback control are also possible.

Because email-control techniques in use at a mail server can send out of band feedback with SMTP, controlling SMTP feedback to senders is not enough to protect recipient's private data. For example, graph-turing tests for ensuring human-initiation of email messages [4] respond to incoming messages with a puzzle that can mostly be solved by humans. Senders can, thus, infer that mail address belongs to a real user and being protected against unwanted mail. This signal also informs the sender that the sent message was able to overcome the recipient's Bayesian filters. This knowledge can further help a malicious sender in propagating unwanted emails in future. Apart from the efficacy of filter rules, a recipient or a domain may wish to protect a lot of other private data, like their email behavior, the set of their email acquaintances, *etc.*

In this chapter, we identify two types of email-control mechanisms, *viz.*, *leaky mechanisms* like monetary bonds, acknowledgement receipts, *etc.*, and *sensitive mechanisms* like white-lists, *i.e.*, the set of senders from whom a recipient always accepts emails, blacklists, *i.e.*, the set of senders from whom the recipient does not wish to receive messages, filters, *etc.* A leaky mechanism is defined as an email-control mechanism that, when used, informs the sender whether his or her message was accepted by the recipient or not. Whereas, a sensitive mechanism is defined as an email control mechanism that uses recipient's private information to decide whether to accept a message or not, but does not disclose any information to the sender. However, if these two types of mechanisms are used in combination, disclosure of recipient's private information is possible. It is the objective of this chapter to prevent such disclosures. Readers may be familiar with leakages due to well-crafted web addresses and images embedded within a message that provide automatic acknowledgement receipts. Later in this chapter we provide details on such leakages. Mechanisms like blacklists, filters, *etc.*, are sensitive because of the nature of the information they control and because their knowledge can help a malicious sender to bypass the control they provide.

The abundance of email-control solutions and the need for automation of several aspects of user's email agents have led to the use of policies that allow flexible control over the behavior of local email systems. Such policies are easily constructed through end user input (*e.g.*, simple user feedback allows Gmail to display or not display embedded images, *etc.*) and through explicit administrator level policies, leading to considerable automation of repetitive tasks. However, because the email system is highly automated, there exists a potential for confidential information to be leaked unintentionally. Even though it is not guaranteed that using a means to leak information will reveal information, however, the probability of leakage of sensitive information, when using leaky and sensitive mechanisms in combination, is non-zero. In particular, schemes that allow sharing acceptance policies to stop undesirable messages earlier in the transmission process (see [55]) compound the problem. Armed with this knowledge, an attacker can simply send a large volume of messages and extract sensitive information from the behavior of the feedback channel.

Our model attacker assumes basic capabilities of computing unfold/fold transformations [56], computing Clark completion of predicate definitions, and the ability to generate a large number of messages. In the worst case analysis, the attacker need send only a $O(n)$ number of message, where n is the size of the policy. Assuming that the private data is not explicitly disclosed to the attacker, we suggest two program transformation techniques: the *necessary policy transformation* and the *sufficient policy transformation* that can be used in tandem to prevent leakages, while leakage channels are still active. We show that these policies are semantically closest to the original policy, while preventing leakages.

6.1.1 Overview of our approach

A survey of recent proposals and initiatives for controlling unwanted messages give sufficient evidence of an eventual move towards policy-controlled email systems. Existing implementations exhibit varied policy evaluation strategies, from complete secrecy (like silent dropping of messages identified as unwanted during Bayesian filtering [30]) to requests for additional information (like human verification tests [4]). Bringing all strategies under a single umbrella enables, both, sharing and hiding of acceptance criteria. Clearly, sanitization of acceptance policies is a prerequisite to communicating them upstream.

Our first step in policy sanitization is to distinguish leaky mechanisms and sensitive information in the policy syntax. Next, we provide a syntactic transformation of the original policy into two zero-information leakage policies and show that they don't leak protected information. The first transformation simply drops all references to sensitive information. The resultant policy, called necessary policy, identifies a set of criteria thus must be satisfied, assuming best case scenario with respect to sensitive information. Similarly, the second transformation constructs a sufficient policy that assumes worst case scenario with respect to sensitive information and identifies messages that can still be accepted. The necessary policy can be shared without risk of leakages, while sufficient policy is designed to be applied at only at the recipient end – thereby achieving complete secrecy in policy evaluation.

Protection against disclosure is a standard problem that has been previously studied in many areas, for example, protection of sensitive information in database transactions (Pfleeger [57], Chapter 6). We analyze the problem in the context of emails, which is very different from other application domains where this problem has been studied. Next we survey some of the disclosure solutions and argue why they are different from our domain.

6.2 Examples

We focus on automatic leakage of information through the email system. Several types of information may be regarded as valuable by different classes of message senders. For example, a large set of valid email users or the strength of message filtering rules of an email domain would be valuable to bulk emailers. For these and other reasons senders may want to know if their messages were read by the recipient, even if the recipient does not wish to release an acknowledgment receipt. We provide some basic examples below how the system could be manipulated to yield such confirmations.

6.2.1 Direct disclosure

SMTP, the default email protocol, allows leakage of information, as discussed earlier. In table 6.1 we list some of the reply codes that can be used for gaining confirmation of valid/invalid email addresses and is an example of direct leakage. In addition, email-control schemes using protocols layered on top of the SMTP protocol can also result in leakage of information. For example, graph-turing tests [4] generate a human-solvable challenge for incoming messages, and accept messages only if the answer is correct. However, issuing a challenge confirms that the recipient address is in use. As these disclosures are made through feedback provided in the protocol, they can be prevented by modifying the behavior of SMTP state machine. In the rest of the chapter, we assume that these disclosures can be prevented using policy-based control schemes for feedback control as discussed in previous chapters.

Table 6.1: Leakage through SMTP reply codes

Code	Meaning	Confirms
251	User not local; will forward to ⟨email address⟩	Forwarding address
450	Mailbox unavailable	Invalid address
452	Insufficient system storage	Valid address
550	Mailbox unavailable	Invalid address
551	User not local; try ⟨email address⟩	Forwarding address
553	Mailbox name not allowed	Invalid address

6.2.2 Disclosure through leaky mechanisms

Well-crafted URLs or images in a message are a prime example of how malicious senders generate acknowledgement receipts without requiring any recipient action. Such a message when viewed or the URL visited by the recipient can cause HTTP requests to a web server that confirms that the recipient read the message. Prevention of automatic acknowledgements is possible through policy based control over the type of messages that can contain HTTP content. Though URLs and images don't qualify as a leaky mechanism (because they are not email-control mechanisms), similar leakages are possible through other mechanisms. For example, bonds inherently leak information irrespective of whether feedback is provided by the email system itself or not. This is because seizure of bond causes monetary flow and therefore informs the person posting that bond that the recipient read the message. This can help a sender infer certain information about the recipient email system as well as recipient's private information. We characterize these leakages as follows:

- **Confirmation of email address:** Confirmation of email addresses is desired (usually by bulk emailers) for increased 'viewership'.

- **Leakage of sensitive information:** Validity of address is known by sender; additional private information, like contents of filter rules, reputation lists are sought.

As an example of the second case, we consider a simple example next to illustrate the basics of an attack.

Example 9 (Leakage through monetary bonds)**.** *Consider a simple recipient policy that allows messages from people not on her blacklist if they attach a bond valued at least at \$ a; for all other users, a bond worth \$ b (b > a) is required. Such a scenario is easily foreseen as monetary signalling techniques ([24, 3, 58, 29], etc.) have recently been applied (in various capacities) in real email networks. We represent this policy informally next. A formal definition of syntax is presented later.*

$$\textit{accept} \quad -if- \quad \textit{some 'allow' rule is true and all 'disallow' rules are false} \tag{6.1}$$

$$\textit{allow} \quad -if- \quad \textit{sender is not blacklisted and message is bonded with value } a \tag{6.2}$$

$$\textit{allow} \quad -if- \quad \textit{for all other senders message is bonded with value } b > a \tag{6.3}$$

$$\textit{disallow} \quad -if- \quad \textit{if message has an attachment with extension .scr} \tag{6.4}$$

Suppose a sender knows that the recipient is using a policy based on bond values and blacklists. Further, because bond seizure confirms if a message was read, the sender can guess values of a, b and whether a sender is on the recipient's blacklist, and verify these guesses by sending a large number of messages while observing the feedback channel. With this information, the sender can easily verify if an email address he can send mail from is in the recipient's blacklist or not – by sending as little as only one email message with bond of value \$ c, c ∈ (a,b) attached. Assuming that the targeted recipient seizes bonds for all commercial mail delivered, no seizure or seizure of bond will prove to the sender that he is *not on the blacklist or not, respectively.*

6.3 Formal Model

A formal model for a policy based decision on email acceptance was presented earlier [55]. We next discuss a constraint logic programming (CLP) based syntax where sensitive and leaky mechanisms are modeled by *private* and *sensitive* predicates, respectively. In particular, we assume that each message is evaluated by a single acceptance policy instead of multiple policies authored by different principals in an email pipeline [55]. As our syntax is more general, it can be specialized to represent any of the policies suggested earlier, or their composition.

6.3.1 Syntax

Logical elements like terms, constraints, *etc.*, are usually defined (please refer to chapter 4 definitions 1 –10). Here we define the sensitive and private predicates and their transformations.

Definition 32 (Predicates)**.** *Predicate symbols are partitioned into three sets: R_D, which are the user defined predicates, R_U, which are the system defined predicates, and R_A is the set of predicates that are guesses for predicates in R_D. In particular, we assume that predicate symbols* allow *and* disallow $\in R_D$ *and* accept $\in R_U$. .

Definition 33 (Private and Sensitive Predicates)**.** *Subsets of R_D predicates, represented by $\mathcal{P}$ and $\mathcal{L}$, form the set of private and sensitive predicates, respectively.*

Definition 34 (System-defined Predicates R_U)**.** *R_U predicates are further partitioned into following sets:*

Mch *For each predicate $p_i \in \mathcal{P}$, two predicate symbols, matchP$_i$ and matchNotP$_i$ of same arity as p_i, are reserved to be defined by the program. In addition for every predicate $Q_j \notin \mathcal{P}$, the program reserves predicate symbols $Q_jMatchP_i$ and $Q_jMatchNotP_i$*

Pes *For every predicate Q, such that $Q \notin \mathcal{P}$, the program reserves a predicate symbol 'pesQ', Q's pessimistic version (defined in section 6.5).*

Opt *For every predicate Q, such that $Q \notin \mathcal{P}$, the program reserves a predicate symbol 'optQ', Q's optimistic version (defined in section 6.5).*

Definition 35 (Atom and Literal)**.** *An atom is of the form $q(t_1, \ldots, t_n)$ where q is a symbol from $R_D \cup R_U \cup \{=, \neq, \leq, \geq\}$ and $t_1, \ldots, t_n$ are terms. A literal is an atom (called a positive literal) or its negation (called a negative literal).*

Definition 36 (Clause, Fact and Rule)**.** *A clause is of the form $H \leftarrow B$ where H is an atom, and B is a list of literals. A fact is a clause in which B is an empty list or a list of literals with predicate symbols from the set $\{=, \neq, \leq, \geq\}$. A clause is called a rule otherwise.*

Definition 37 (CLP Program)**.** *A CLP Program (simply a program) is a set of clauses. For a program Π and a predicate P, $P \propto \Pi$ if for any rule $H \leftarrow B_1,\ldots,B_n$ in Π, $P = H\theta$ or $P = B_i\theta$ ($i \in [1,n]$) for some θ.*

Definition 38 (Message)**.** *A message is a set of facts*

Definition 39 (Mail Acceptance Policy)**.** *A mail acceptance policy, or simply, a policy is a pair $\Pi = \langle \Pi_R, \Pi_D \rangle$ where Π_R is a set of rules (ruleset) and Π_D is a set of facts. The program Π_R is required to be stratified and contain definitions of top level predicate accept and at least one of the predicates: allow, disallow. The predicate symbol accept is always* *defined as*

$$accept(\overrightarrow{msg}) \leftarrow allow(\overrightarrow{msg}), \neg disallow(\overrightarrow{msg})$$

Here $\overrightarrow{msg}$ tuple represents all the variables and their bindings derived from a message, *e.g.*, *From, To, Time, Bond*, *etc.*, would all be included in the tuple. Predicates other than *allow, disallow* may have single variables from $\overrightarrow{msg}$ tuple as arguments.

6.3.2 Semantics

We reuse the three-valued semantics (with constructive negation) used in [55], and chapters 4,5. A message is accepted if c| $accept(\overrightarrow{msg}) \in T_P^+ \uparrow \omega$ where $\overrightarrow{msg}$ is a tuple of headers and content supplied in the message. The authors show that the decision procedure using the presented semantics is complete [55]. Finally we define the extension and Clark completion of a predicate as follows.

Definition 40 (Extension of a predicate)**.** *Extension of a predicate p is the set ext(p)* $\subset T_P^+(I)$ *such that each constrained atom in ext(p) is of the form c|* $p(\overrightarrow{x})$

Definition 41 (Clark completion)**.** *Given a ruleset* Π*, each predicate p,* $p \propto \Pi$ *such that for some rule* $\pi \in \Pi$ $p(\overrightarrow{x}) = head(\pi)$*, is associated with a logical formula as follows. If there are n rules in* Π*:*

$$\begin{aligned} p(\overrightarrow{x}) &\leftarrow B_1 \\ &\vdots \\ p(\overrightarrow{x}) &\leftarrow B_n \end{aligned}$$

then the formula associated with p is

$$\begin{aligned}\forall \overrightarrow{x}\ p(\overrightarrow{x}) \ \leftrightarrow \quad & \exists \overrightarrow{y_1}\ B_1 \\ & \vee \exists \overrightarrow{y_2}\ B_2 \\ & \vdots \\ & \vee \exists \overrightarrow{y_n}\ B_n\end{aligned}$$

where $\overrightarrow{y_i}$ *is the set of variables in* B_i *except for variables in* $\overrightarrow{x_i}$*. If* $p \neq head(\pi)$ *for any* $\pi \in \Pi$*, then the formula associated is*

$$\forall \overrightarrow{x} \neg p(\overrightarrow{x})$$

The collection of all such formulas is called the Clark completion of Π*. We represent the Clark completion of a predicate p by* p^*

6.4 The Attacker Model

Next we define the attacker's capabilities and model for leaking private information. An attacker is constrained to legal runs of SMTP protocol. However, the attacker is not restricted to gaining information from the SMTP protocol alone. There is no restriction on the number of email messages an attacker can generate, and these messages can be targeted to any recipient. For worst case analysis we make following assumptions:

1. Policies used at an email domain may be known, *e.g.*, use of blacklists, whitelists, filters, *etc.* This is possible through explicit communication of portions of policies or through other means (like attacker knows about the victim's policy by

the virtue of being served by the same email service provider, say Hotmail, Gmail *etc.*). In particular Π_R (rule set) may be known but not Π_D (set of facts) that contain the definitions of private predicates.

2. By observing protocol runs alone an attacker cannot conclude if a message was delivered. , *i.e.*, recipient domain may indicate that the message was delivered, without actually delivering the message. Confirmation of message acceptance can be obtained from leaky mechanisms alone.

3. Recipient acts on every delivered message, like, seizes bond for unwanted message, *etc.*

6.4.1 Capabilities

An attacker is assumed to possess a basic set of capabilities that help launch attacks against recipient's private information. For example, an attacker can compute Clark completion of programs or compute 'unfold/fold' type transformations of *normal* CLP programs that preserve the program semantics. Existing literature on unfold/fold transformations show how to preserve different types of semantics by allowing specific variations in their unfold/fold operations. For our normal (stratified) programs and a three-valued semantics of programs, Sato's [59] equivalence-preserving first-order unfold/fold transformations are the best fit. Fages' work on generalization of the s-semantics approach to normal CLP programs and a fixpoint characterization of Kunen's semantics shows that Sato's transformations can be easily supported in our model. However, for simplicity, we describe unfold/fold transformations very briefly here and for additional information, the reader is referred to the original papers [60, 59, 56]. Given a set of rules $\Pi = \{\pi_1, \ldots, \pi_n\}$, and the set P =$\{$q | q $\propto \Pi\}$, an attacker has following capabilities:

1. **Capability of computing Clark completion:** For all $q \in P$, the attacker can compute q^*, q's Clark completion with respect to $\mathcal{P}$.

2. **Capability of unfold transformation [60, 59, 56]:** Given a rule π_k: H $\leftarrow$ A, B, C where A, C $\subset$ P and B $\in$ P is a positive literal such that for some rule π_i and some θ where B = head(π_i)θ, the attacker can transform π_k to H $\leftarrow$ A, body(π_i)θ, C (here *head* and *body* functions map a rule to the atom in its head and literals in its body, respectively, where variables in π_i and π_k are renamed apart. We represent the fully unfolded form of a program Π by Π^ω.

3. **Capability of fold transformation [60, 59, 56]:** Given a rule π_k: H $\leftarrow$ A, B, C where A, B, C $\subset$ P such that for some rule π_i and some θ such that B = body(π_i)θ, the attacker can transform π_k to H $\leftarrow$ A, head(π_i)θ, C

4. **Capability of message generation:** An attacker can generate any number of messages.

The email policy attacker uncovers a subset of private predicates' extension. The security goal of this paper is to prevent an attacker from gaining access to recipient's private information. Computing Clark completion of program definitions, unfold/fold transformations and sending messages are the assumed capabilities

An attack on extension of private predicate involves the sender sending messages (sets of facts like the sender address, recipient address, time of transmission, bond value, *etc.*) that contain specific values for arguments of a leaky predicate. These values are generated by analyzing the recipient's policy. The attacker is made known of the fact that c|accept($\overrightarrow{m}$) belongs to $T_P^+(I)$ or $T_P^-(I)$ through the signals received from leaky mechanism. With these information, the attacker can construct ext(p′) $\subset$ ext(p), where p is a private predicate and p′ $\in R_A$.

6.4.2 Scripting an attack

Next we show how to attack using capabilities defined above.

1. Given a set of rules Π such that *accept* $\propto \Pi$, the attacker computes Π^ω, the fully unfolded form of Π (the head p of each fully unfolded rule is referred to as p^ω). This operation yields clauses with *accept*$^\omega$ heads, the fully unfolded form of rules with *accept* as their head.

2. In the next step the attacker constructs the Clark completion of *accept*$^\omega$ to yield *accept*$^{\omega *}$.

3. Using *accept*$^{\omega *}$, the attacker can then generate guesses by analyzing the values of leaky mechanism that can generate messages to verify her guess.

The unfold/fold transformation belongs to NP complexity class [61], as does the Clark completion operation. Overall, the complexity of policy attack is NP.

Example 10. *We provide the formal syntax of policy in example 9 and show how an attack can be orchestrated against it. Here,* blacklist *is a private predicate, whose definition (or extension) is hidden from the attacker and* $atrb_{\text{bond}}$ *is a leaky predicate. The rules (2), (3)* *and (4) from example 9 written in the above syntax are as follows:*

$$\begin{aligned} allow(\overrightarrow{m}) &\leftarrow \neg blacklist(Y), atrb_{\text{bond}}(X), X \geq 5 \\ allow(\overrightarrow{m}) &\leftarrow blacklist(Y), atrb_{\text{bond}}(X), X \geq 10 \\ disallow(\overrightarrow{m}) &\leftarrow atrb_{\text{ext}}(`.scr') \end{aligned}$$

Similarly, rule (1) in example 9 is encoded as:

$$accept(\overrightarrow{m}) \leftarrow allow(\overrightarrow{m}), \neg disallow(\overrightarrow{m})$$

Using the unfolding capability, accept predicate definitions can be transformed to (for simplicity, we ignore disallow clause):

$$\begin{aligned} accept(\overrightarrow{m}) &\leftarrow \neg blacklist(Y), atrb_{\text{bond}}(X), X \geq 5 \\ accept(\overrightarrow{m}) &\leftarrow blacklist(Y), atrb_{\text{bond}}(X), X \geq 10 \end{aligned}$$

Next the attacker can compute Clark completion of accept definition:

$$\begin{aligned} \forall \overrightarrow{m}\ accept(\overrightarrow{m})^* \leftrightarrow\ & \exists Y_1, X_1\ \neg blacklist(Y_1), atrb_{\text{bond}}(X_1), X_1 \geq 5 \\ & \vee \\ & \exists Y_2, X_2\ blacklist(Y_2), atrb_{\text{bond}}(X_2), X_2 \geq 10 \end{aligned}$$

An attacker is now in a position to guess parts of the extension of blacklist using following rule:

$$\begin{aligned} blacklist'(Y_g) \leftarrow\ & \neg accept(\overrightarrow{m_1}), accept(\overrightarrow{m_2}), atrb_{\text{bond}}(X_1), \\ & atrb_{\text{bond}}(X_2), X_1 \in [5, 10], X_2 > 10 \end{aligned}$$

Here blacklist′ $\in R_A$, is defined by the attacker. The attacker can send two messages with all facts same except the bond values. The first message (m_1) is bonded with a value $v \in$

(5,10) and second one (m_2) bonded with a value greater than 10. It is easy to see that if $Y_g \in ext(blacklist)$, then the sender will get one negative and one positive verification – $c|accept(\overrightarrow{m_1}) \in T_P^-(I)$ and $c|accept(\overrightarrow{m_2}) \in T_P^+(I)$; otherwise both verifiers are positive.

6.5 Policy transformations for privacy

To prevent an attacker from deducing subsets of recipient maintained set(s) of private information, we propose to transform the evaluation policy such that leakage signals are rendered useless. There are two flavors of transformation that we propose: *the sufficient policy* and *the necessary policy* transformation. Intuitively, the sufficient policy should accept a message just in case the message is accepted by the original policy under *all* possible definitions of the private predicates. On the other hand, the necessary policy accepts a message for *some* definition of the private predicates in the original policy, thereby ensuring that only messages satisfying the necessary policy can satisfy the original policy. These policies are designed to be used in tandem, *i.e.*, single evaluation of original policy is replaced by the evaluation of necessary and sufficient policies.

6.5.1 Transformation algorithm

Transformation algorithm is discussed next. Because only those rules that use private literals in their bodies can leak private information, the algorithm applies to such rules and leaves others unchanged. The transformation algorithm is shown in figure 6.1 and consists of two transformations for each rule containing sensitive predicates and is described in detail next.

Figure 6.1 begins with a general Horn clause representation of rules in Π_R with meta-variables Q$_u$, Q$_v$ and p$_k$ and $\overrightarrow{m}$ is the tuple of all variables used in Π_R. $Q_u(\overrightarrow{y})$ represents a non-sensitive literal at the u^{th} position in a rule, and can also appear in the head of the rule. The rule is shown to have v non-sensitive predicates in its body and some sensitive predicates p_k, for $k \in [1,\, t^{'}]$, each used positively m_k times and negatively n_k times. In other words, recursive calls and multiple calls to the same predicate may be made in a rule, *i.e.*, Q_u may be in $[Q_1,\, Q_v]$ or $Q_{u_1} = Q_{u_2}$ for u_1, u_2 $\in [1,\, v]$, $u_1 \neq u_2$. However, Q_u cannot make recursive calls to itself through negation or include calls such that the program dependency graph includes negative cycles due to the stratification. Also, each literal p_k need not appear in the body of every clause Q_u, *i.e.*, both m_k and n_k can be equal to zero.

As shown in the figure, each Q_u definition is transformed to two related predicates, *viz.*, *pes*Q_u and *opt*Q_u, where *pes*Q_u is the 'pessimistic' version of Q_u, independent of the definition of any private predicate used in the definition of Q_u, and *opt*Q_u is the 'optimistic' version of Q_u predicate, which holds for 'some' definition of private predicates. More precisely, *opt*Q_u will hold if there exists *some* definition of private predicates used in the definition of Q_u, such that Q_u can be shown to hold in Π, whereas *pes*Q_u will only hold if for all definitions of private predicates, Q_u can be shown to hold true in Π.

It must be noted that the algorithm, as presented, does not include the details of how transformed and non transformed rules are linked. Suppose there is a predicate $Q(\overrightarrow{x})$ in the body of a transformed clause that does not use any sensitive literals. The transformation still renames it as *pes*$Q(\overrightarrow{x})$ whenever it is used positively, and *opt*$Q(\overrightarrow{x})$ when it is used negatively. However, the transformed versions of the definition of $Q(\overrightarrow{x})$ are not created since it does not use any sensitive predicates in the body. Hence we add two rules for each such predicate, which are, $pesQ(\overrightarrow{x}) \leftarrow Q(\overrightarrow{x})$ and $optQ(\overrightarrow{x}) \leftarrow Q(\overrightarrow{x})$. In example 11 we present a concrete example of this transformation.

Example 11. *Pessimistic and optimistic transformations.*

Consider the Π_R definition of predicate trusted(x,...,z) that uses non sensitive predicates professor(Profile), student(Profile) and bonded(B, minValue) and private predicate blacklist(X_{From}) defined in Π_D (ignore the distinction between 'atrb' and other predicates):

$$\begin{aligned}
trusted(\overrightarrow{x}) &\leftarrow professor(X_{\mathit{From}}) \\
trusted(\overrightarrow{x}) &\leftarrow student(X_{\mathit{From}}), \neg blacklist(X_{\mathit{From}}) \\
trusted(\overrightarrow{x}) &\leftarrow blacklist(X_{\mathit{From}}), bonded(X_{\mathit{X\text{-}Bnd}}, 5)
\end{aligned}$$

The optimistic and pessimistic forms of the predicate trusted in Π_{suf} are as follows. For simplicity we retain the names of other predicates (i.e., student, professor, bonded are unchanged), however, in reality, their pessimistic and optimistic versions coincide. Also, we use trustedMB symbol for trustedMatchBlacklist and trustedMNB for trustedMatchNot-Blacklist predicate due to space constraints:

$$\begin{aligned}
pesTrusted(\overrightarrow{x}) &\leftarrow professor(X_{\mathit{From}}) \\
pesTrusted(\overrightarrow{x}) &\leftarrow trustedMB(\overrightarrow{y_1}), trustedMNB(\overrightarrow{y_2}) \\
trustedMB(\overrightarrow{y_1}) &\leftarrow student(X_{\mathit{From}}) \\
trustedMNB(\overrightarrow{y_2}) &\leftarrow bonded(X_{\mathit{X\text{-}Bnd}}, 5) \\
optTrusted(\overrightarrow{x}) &\leftarrow student(X_{\mathit{From}}) \\
optTrusted(\overrightarrow{x}) &\leftarrow bonded(X_{\mathit{X\text{-}Bnd}}, 5)
\end{aligned}$$

Computational Complexity We next relate the size of the transformed ruleset to the size of the original ruleset. The algorithm distinguishes between two types of clauses: those that use private predicates in their body and those that do not, referred to as π_1 and π_2 respectively. Clauses that do not contain private predicates in the body (*i.e.*, the clauses in the π_2 set) are preserved in their original form. However, in the worst case scenario, atoms in the head of each such clause appears in at least one clause that has sensitive predicates in the body. Therefore, for each such clause, two additional clauses are required to 'link' the pessimistic and optimistic versions of the predicate to the original definition. Therefore, the size of the set corresponding to π_2 in Π_R in the transformed policy is $3 \times |\pi_2|$.

For the clauses in the set π_1, the transformation algorithm can potentially construct a much bigger set. First, for each clause in π_1 optimistic and pessimistic versions (Pes and Opt predicates) are created that contain Mch predicates. The number of pessimistic clauses is equal to the number of distinct private predicates in the body of the original clause, whereas only one optimistic predicate is constructed. Next, for each new 'Mch' predicate introduced, a clause is added. Assuming that the maximum number of distinct private predicates used in a clause in π_1 is some number η, the size of the transformed set is $(2 \times \eta + 1) \times |\pi_1|$. Thus the complexity of the algorithm is $O(\eta \times |\Pi_R|)$.

Necessary Policy

Intuitively, the necessary policy, Π_{nec}, strips away sensitive predicates from the original policy. The basic idea is to generate a policy where satisfaction requirements are in terms of non-sensitive literals, while assuming the best possible scenario with respect to the definition of sensitive predicates. This aim is achieved by the following definition of top-level accept predicate (accept$_{nec}(\overrightarrow{msg})$ for clarity) and while example 12 illustrates the basic idea:

$$accept_{nec}(\overrightarrow{m}) \quad \leftarrow \quad optAllow(\overrightarrow{m}), \neg pesDisallow(\overrightarrow{m})$$

Example 12. *[Illustration of necessary policy] Consider a ruleset Π_R where B_1 and B_2 are a list of positive literals with no literal belonging to $\mathcal{P}$. Hence their 'opt' and 'pes' versions coincide. Also, $p \in \mathcal{P}$*

$$allow(\overrightarrow{msg}) \quad \leftarrow \quad B_1, p(X) \tag{6.5}$$

$$allow(\overrightarrow{msg}) \quad \leftarrow \quad B_2, \neg p(X) \tag{6.6}$$

Applying the necessary transformation we get:

$$\begin{aligned} accept_{nec}(\overrightarrow{m}) \quad &\leftarrow \quad optAllow(\overrightarrow{m}), \neg pesDisallow(\overrightarrow{m}) \\ optAllow(\overrightarrow{m}) \quad &\leftarrow \quad B_1 \\ optAllow(\overrightarrow{m}) \quad &\leftarrow \quad B_2 \end{aligned}$$

By unfolding and completing the definition of $accept_{nec}$ we get ($\overrightarrow{y_1}$ and $\overrightarrow{y_2}$ are free variables in B_1 and B_2 respectively)

$$\forall \overrightarrow{m} \; accept^{\omega *}_{nec}(\overrightarrow{m}) \leftrightarrow \exists \overrightarrow{y_1} \; B_1 \vee \exists \overrightarrow{y_2} \; B_2$$

This policy accepts messages depending upon the clauses of the original policy, with the change that sensitive predicate is dropped from rules 6.5,6.6

Sufficient Policy

The basic idea behind this transformation is to syntactically match the uses of sensitive literals in the body of rules with *allow* head, *e.g.*, use $pesAllow(\overrightarrow{m})$ in place of allow($\overrightarrow{m}$). In other words, we wish to *resolve away* the uses of sensitive literals, akin to the predicate elimination strategy proposed by Reiter [62]. The following top-level predicate accept (accept$_{suf}$ for clarity) achieves this aim:

$$accept_{suf}(\overrightarrow{m}) \quad \leftarrow \quad pesAllow(\overrightarrow{m}), \neg optDisallow(\overrightarrow{m})$$

Example 13 (Illustration of sufficient policy)**.** *Consider the ruleset given by rules 6.5 and 6.6. The sufficient transformation of rules yields the following ruleset*

$$\begin{aligned} accept_{suf}(\overrightarrow{m}) &\leftarrow pesAllow(\overrightarrow{m}), \neg optDisallow(\overrightarrow{m}) \\ pesAllow(\overrightarrow{m}) &\leftarrow matchP(X), matchNotP(X) \\ matchP(X, \overrightarrow{m}) &\leftarrow B_1 \\ matchNotP(X, \overrightarrow{m}) &\leftarrow B_2 \end{aligned}$$

By unfolding and completing the definition of $accept_{suf}$ we get

$$\forall \overrightarrow{m}\ accept^{\omega *}_{suf}(\overrightarrow{m}) \leftrightarrow \exists \overrightarrow{y_1}, \overrightarrow{y_2}\ B_1, B_2$$

This policy accepts messages that simultaneously satisfy the bodies of clauses 6.5 and 6.6, with private predicate stripped off from the rules.

6.5.2 Syntactic Properties

The syntactic properties of necessary and sufficient policies essentially state that the predicates identified as private in the original policy do not occur in transformed policies. These follow in a straightforward manner from the transformation algorithm.

Lemma 3. *Given $P \subseteq \mathcal{P}$ such that if $p_i \in P$ and $p_i \propto \Pi_R$ then $p_i \not\propto \Pi_{nec}$ (resp. Π_{suf}) where Π_{nec} (resp. Π_{suf}) is necessary (resp. sufficient) transformation of Π_R.*

Corollory 1. *Given $P \subseteq \mathcal{P}$ such that if $p_i \in P$ and $p_i \propto \Pi_R$ then $p^{\omega*}$ or p do not occur in $\Pi^{\omega*}_{nec}$ (resp. Π^{*}_{suf}).*

6.5.3 Semantic Properties

To prove how evaluation of Π_{nec} and Π_{suf} instead of Π_R prevents sensitive leakages, we need to show some semantic properties of the transformed rulesets. The program corresponding to the original policy is represented by P, where P $= \Pi_R \cup \Pi_D \cup M$, in which M is a set of message facts, Π_D is the set of private facts and Π_R is a ruleset. We are interested in two forms of P for the purposes of the proof below. The first form is one where we are interested in satisfaction of clauses in Π_R for all definitions of private predicates (Π_{suf}). The second form is one where we are interested in satisfaction of the clauses in Π_R for some definition of the private predicates (Π_{nec}). Assuming Π_D contains only facts constructed from private predicates, we denote the program corresponding to Π_{suf} by P_S, where $P_S = \Pi_{suf} \cup M$ and the program corresponding to Π_{nec} by P_N, where $P_N = \Pi_{nec} \cup M$. Both these programs are independent of the definitions of the sensitive predicates.

We give a general relation between 'optimistic' and 'pessimistic' versions of a literal and the literal in theorem 4. Next we proceed to define the relation between the programmed policy $\Pi_R \cup \Pi_D$ and the generated policies Π_{suf} and Π_{nec}.

'pesQ', 'optQ' *vs.* **'Q'** We begin by relating the satisfaction of 'pessimistic' and 'optimistic' versions to the satisfaction of the original predicate. Intuitively, this means that whenever the pessimistic version of a predicate is true, then the original predicate is also true, irrespective of the truth values of the sensitive predicates. Similarly, 'optimistic' version being satisfied implies that there is a possible definition of private predicates (in the set of program facts, Π_D), such that the original predicate is satisfied.

Theorem 4. *Given a program $P = \Pi_R \cup \Pi_D \cup M$, in which $\Pi_R \cup \Pi_D$ is a policy that includes sensitive predicates p_1 to p_t defined in Π_D and M is a set of facts, any literal $pesQ_u(\overrightarrow{y})$ in the program $P_S = \Pi_{suf} \cup M$ or $P_N = \Pi_{nec} \cup M$, apart from the $accept(\overrightarrow{msg})$ atom, is satisfied if and only if for all definitions of $p_1, \ldots, p_t$, $Q_u(\overrightarrow{y})$ is satisfied in P, and $optQ_u(\overrightarrow{y})$ is satisfied if and only if there exists some definition of $p_1, \ldots, p_t$ such that $Q_u(\overrightarrow{y})$ is satisfied.*

Proof:

In all the three programs, $Q(\overrightarrow{y})$ is true if $c \; |Q(\overrightarrow{y}) \in T^+_P{\uparrow}\omega$. Because both P_S and P_N have exactly similar definitions of all the literals except the $accept(\overrightarrow{msg})$ atom, hence, in the rest of the proof, we only show the result for P_S, and the same result for P_N is implied.

($\Rightarrow$) We show the $pesQ_u(\overrightarrow{y})$ part of the proof by induction on the steps of T_{P_S} construction. The induction hypothesis has four parts. First part states that given $c \mid p_k(\overrightarrow{x}) \in T^-_P \uparrow \alpha$, $c \mid Q_u \; matchP_k(\overrightarrow{x}, \overrightarrow{m}) \in T^+_{P_S} \uparrow \alpha$ entails $c \mid Q_u(\overrightarrow{x}) \in T^+_P{\uparrow}\beta$ for some β. Similarly, it states that given $c \mid p_k(\overrightarrow{x}) \in T^+_P \uparrow \alpha$, $c \mid Q_u \; matchNotP_k(\overrightarrow{x}, \overrightarrow{m}) \in T^+_{P_S} \uparrow \alpha$ entails $c \mid Q_u(\overrightarrow{x}) \in T^+_P \uparrow \beta$ for some β. The third part states that $c \mid optQ_u(\overrightarrow{x}) \in T^-_{P_S} \uparrow \alpha$ entails $c \mid Q_u(\overrightarrow{x}) \in \uparrow \beta$ for some β and the fourth part states that $c \mid pesQ_u(\overrightarrow{x})$ in $T^+_{P_S}\!\uparrow \alpha$ entails $c \mid Q_u(\overrightarrow{x}) \in T^+_P \uparrow \beta$.

For the base case, $T^+_{P_S} \uparrow 0 = \emptyset$, and the hypothesis trivially holds. For the successor ordinals, we assume that the hypothesis holds, hence the atoms added in the step α, *i.e.*, $c \mid Q_u matchP_k(\overrightarrow{x}, \overrightarrow{m})$, $c \mid Q_u matchNotP_k(\overrightarrow{x}, \overrightarrow{m})$, $c \mid pesQ_u(\overrightarrow{x})$ and $c \mid optQ_u(\overrightarrow{x}) \in T^+_{P_S} \uparrow \alpha$. Next consider $T^+_{P_S} \uparrow \alpha{+}1$. We wish to show that the induction hypothesis holds for this step as well. First consider the general form of a $Q_u matchP_k(\overrightarrow{x}, \overrightarrow{m})\theta$ atom that is added in this step:

$$\begin{aligned}
Q_u matchP_k(\overrightarrow{x^k_{m_k+j}}, \overrightarrow{m}) \;\leftarrow\;& pesQ_1(\overrightarrow{y_1}), \ldots, pesQ_o(\overrightarrow{y_o}), \neg optQ_{o+1}(\overrightarrow{y_{o+1}}), \ldots, \\
& \neg optQ_{o+s}(\overrightarrow{y_{o+s}}), Q_u matchP_1(\overrightarrow{x_{1,1}}, \overrightarrow{m}), \ldots, \\
& Q_u matchP_k(\overrightarrow{x_{m_k,k}}, \overrightarrow{m}), Q_u matchNotP_k(\overrightarrow{x_{m_k+1,k}}, \overrightarrow{m}), \\
& \ldots, Q_u matchNotP_k(\overrightarrow{x_{m_k+(j-1),k}}, \overrightarrow{m}), \\
& Q_u matchNotP_k(\overrightarrow{x_{m_k+(j+1),k}}, \overrightarrow{m}), \ldots, \\
& Q_u matchNotP_t(\overrightarrow{x_{m_t+n_t,t}}, \overrightarrow{m}), \overrightarrow{x_{i,k'}} \neq \overrightarrow{x_{m'_k+j,k'}}, \\
& i \in [1, m'_k], j \in [1, n'_k], k' \in [1, t], c.
\end{aligned}$$

The above clause is derived from the following clause in Π:

$$\begin{aligned}
Q_u(\overrightarrow{x}) \;\leftarrow\;& Q_1(\overrightarrow{y_1}), \ldots, Q_o(\overrightarrow{y_o}), \neg Q_{o+1}(\overrightarrow{y_{o+1}}), \ldots, \neg Q_{o+s}(\overrightarrow{y_{o+s}}), p_1(\overrightarrow{x_{1,1}}), \ldots, \\
& p_1(\overrightarrow{x_{m_1,1}}), \neg p_1(\overrightarrow{x_{m_1+1,1}}), \ldots, \neg p_1(\overrightarrow{x_{m_1+n_1,1}}), \ldots, p_k(\overrightarrow{x_{1,k}}), \ldots, \\
& p_k(\overrightarrow{x_{m_k,k}}), \neg p_k(\overrightarrow{x_{m_k+1,k}}), \ldots, \neg p_k(\overrightarrow{x_{m_k+n_k,k}}), \ldots, p_t(\overrightarrow{x_{1,t}}), \ldots, \\
& p_t(\overrightarrow{x_{m_t,t}}), \neg p_t(\overrightarrow{x_{m_t+1,t}}), \ldots, \neg p_t(\overrightarrow{x_{m_t+n_t,t}}), c.
\end{aligned}$$

We wish to show that this clause generates $c \mid Q_u(\overrightarrow{x}) \in T_P^+ \uparrow \beta$ for some β. Because each *pesQ* constrained atom corresponding to the literal in the body of $Q_u match P_k(\overrightarrow{x}, \overrightarrow{m})$ must be in $T_{P_S}^+ \uparrow \alpha$ and each *optQ* constrained atom in $T_{P_S}^- \uparrow \alpha$, by inductive hypothesis each constrained atom of positive Q literal in the body of Q_u clause is in $T_P^+ \uparrow \gamma$ and each c.atom of negative literal is in $T_P^- \uparrow \gamma$ (since membership of entails membership of T_P). There are three cases to consider based on Π_D. In the first case, if all p_k literals in the body of $Q_u(\overrightarrow{x})$ evaluate to true, then $c \mid Q_u(\overrightarrow{x})$ $\in T_P^+ \uparrow \gamma + 1$ since the rest of the literals allow such a deduction, as shown in the bottom-up semantics presented earlier. In the second case, if *some* $c \mid p_k(\overrightarrow{x_{i,k}}) \in T_P^- \uparrow \gamma$, *i.e.*, if some sensitive literal used positively in $Q_u(\overrightarrow{x})$ definition is in T_P^-, then the corresponding constrained atom $c \mid Q_u match P_k(\overrightarrow{x_{i,k}}, \overrightarrow{m})$, which must be in $T_{P_S}^+ \uparrow \alpha$, entails $c \mid Q_u(\overrightarrow{x}) \in T_P^+ \uparrow \gamma + 1$ due to the inductive hypothesis. Similarly, in the third case, if *some* negatively used literals $c \mid p_k(\overrightarrow{x_{m_k+j,k}}) \in T_P^+ \uparrow \gamma + 1$, then the corresponding $c \mid Q_u match Not P_k(\overrightarrow{x_{m_k+j,k}}, \overrightarrow{m})$ constrained atom in $T_{P_S}^+ \uparrow \gamma$ entails $c \mid Q_u(\overrightarrow{x}) \in T_P^+ \uparrow \beta$. Therefore, in every case it can be shown that $c \mid Q_u(\overrightarrow{x}) \in T_P^+ \uparrow \beta$.

It is straightforward to see that the second part of the inductive hypothesis can be shown for $c \mid Q_u match Not P_k(\overrightarrow{x}, \overrightarrow{m})$ constrained atom added in $T_{P_S}^+$ under the assumption that $c \mid p_k(\overrightarrow{x}) \in T_P^+ \uparrow \gamma$ based on the proof of the first part. We show the fourth part of the inductive hypothesis next. As every *pesQ* literal is defined as $pesQ_u(\overrightarrow{x}) \leftarrow Q_u match P_k(\overrightarrow{x}, \overrightarrow{m}),\ Q_u match Not P_k(\overrightarrow{x}, \overrightarrow{m})$ therefore, $c \mid pesQ_u(\overrightarrow{x})$ in $T_{P_S} \uparrow \alpha + 1$ implies $c \mid Q_u(\overrightarrow{x})$ in $T_P^+ \uparrow \gamma$.

Lastly, we need to show the third part of the inductive hypothesis, *i.e.*, $c \mid optQ_u(\overrightarrow{x})$ $\in T_{P_S}^- \uparrow \alpha + 1$ entails $c \mid Q_u(\overrightarrow{x}) \in \uparrow \gamma$. Because $c \mid optQ_u(\overrightarrow{x})$ is in $T_{P_S}^- \uparrow \alpha + 1$, in all its definitions, constrained form of some literal in its body, by the virtue of its membership of $T^-_{P_S}\uparrow\alpha$, always prohibits $c\ |optQ_u(\overrightarrow{x})$ literal's addition to $T^+_{P_S}\uparrow\alpha+1$. By the inductive hypothesis, it can be easily shown that either for some $k \in [1,\, o]$, $c\ |Q_k(\overrightarrow{x_k})$ $\in\ \uparrow\gamma$ or for some $k \in [o,\, s]$, $c\ |Q_k(\overrightarrow{x_k}) \in T^+_P\uparrow\gamma$ for every clause defining $Q_u(\overrightarrow{x})$.

For the limit ordinal case, $c\ |matchP_k(\overrightarrow{x},\overrightarrow{m}) \in T^+_{P_S}\uparrow\alpha$ (respectively, constrained atoms $c\ |\ matchNotP_k(\overrightarrow{x},\overrightarrow{m})$, $c\ |pesQ_u(\overrightarrow{x})$) entails that the constrained atoms $c\ |\ matchP_k(\overrightarrow{x},\overrightarrow{m}) \in T^+_{P_S}\uparrow\beta$ (respectively, $c\ |\ matchNotP_k(\overrightarrow{x},\overrightarrow{m})$, $c\ |\ pesQ_u(\overrightarrow{x})$) for some $\beta < \alpha$, by construction. Also, $c\ |\ optQ_u(\overrightarrow{x}) \in T^-_{P_S}\uparrow\alpha$ entails $c\ |\ optQ_u(\overrightarrow{x})$ $\in T^-_{P_S}\uparrow\beta$ for some $\beta < \alpha$, by construction. The inductive hypothesis now applies, giving the desired result.

($\Leftarrow$) Here we show that if $c\ |Q_u(\overrightarrow{x}) \in T^+_P\uparrow\omega$, it entails $c\ |pesQ_u(\overrightarrow{x}) \in T^+_{P_S}\uparrow\omega$. If there is a $c\ |Q_u(\overrightarrow{x}) \in T^+_P\uparrow\omega$ that does not contain any private literals in the body, the desired result follows immediately, so we assume otherwise. Consider any minimal collection of $Q_u(\overrightarrow{x})$ clauses in Π that together can be used to show $c\ |Q_u(\overrightarrow{x})$ $\in T^+_P\uparrow\omega$. Consider an instance $p_a(\overrightarrow{x_1})$ of a p_k atom occurring in some definition of $Q_u(\overrightarrow{x})$ literal in this collection, say A, such that $c\ |\mathrm{A} \in T^+_P\uparrow\omega$. We claim there must be a clause, say B, in the collection which contains $\neg p_a(\overrightarrow{x_1})$ such that $c\ |\mathrm{B} \in$ $T^+_P\uparrow\omega$. To see this, we assume otherwise and show that A can be removed from the collection without interfering with the collection's ability to prove that $c\ |Q_u(\overrightarrow{x}) \in$ $T^+_P\uparrow\omega$. This contradicts the assumption of minimality. The key observation in this argument is that if in a set of clauses that do not contain the clause with $p_a(\overrightarrow{x_1})$ there is no occurrence of the literal $\neg p_a(\overrightarrow{x_1})$, then those clauses show $c\ |Q_u(\overrightarrow{x}) \in$ $T^+_{P_S}\uparrow\omega$ for all definitions of private predicates such that they make $p_a(\overrightarrow{x_1})$ false without depending on the truthfulness of $p_a(\overrightarrow{x_1})$, and therefore we get the result for all possible Π_R, *i.e.*, $c\ |Q_u(\overrightarrow{x}) \in T_P^+ \uparrow \omega$. The argument can be easily generalized for a set of private literals p_k occurring in $Q_u(\overrightarrow{x})$ clause's body. Because $pesQ_u(\overrightarrow{x})$ rule in Π_{suf} is constructed by pairing corresponding $matchP$ and $matchNotP$ predicates (*i.e.*, $matchP_k(\overrightarrow{x_k}, \overrightarrow{m})$ and $matchNotP_k(\overrightarrow{x_k}, \overrightarrow{m})$), it is straightforward to see that given $c\ |Q_u(\overrightarrow{x}) \in T_P^+ \uparrow \omega$, it entails $c\ |pesQ_u(\overrightarrow{x}) \in T_{P_S}^+ \uparrow \omega$. When there are additional uses of p_k, the argument can be repeated and the recursive definition of *match* atoms used to show that additional clauses are incorporated into the derivation in Π_{suf}.

($\Rightarrow$) For the $optQ_u(\overrightarrow{x})$ part of the proof, we want to show that $c\ |optQ_u(\overrightarrow{x}) \in T_{P_S}^+ \uparrow \omega$ entails $c\ |Q_u(\overrightarrow{y}) \in \uparrow \omega$. We again use induction on the steps of T_{P_S} construction. The inductive hypothesis states that $c\ |optQ_u(\overrightarrow{x}) \in T_{P_S}^+ \uparrow \alpha$ entails $c\ |Q_u(\overrightarrow{x}) \in \uparrow \beta$ for some β. For the base case, $T_{P_S} \uparrow 0 = \emptyset$, hence the hypothesis is trivially satisfied. For the successor ordinals, we assume that the hypothesis holds for any atoms added in the step α, *i.e.*, $c\ |optQ_u(\overrightarrow{x}) \in T_{P_S}^+ \uparrow \alpha$ entails $c\ |Q_u(\overrightarrow{x}) \in \uparrow \beta$, for some β. Next consider the $\alpha + 1$ step and an $c\ |optQ_u(\overrightarrow{x})$ added in the this step:

$$\begin{aligned} optQ_u(\overrightarrow{x}) \quad \leftarrow \quad & optQ_1(\overrightarrow{y_1}), \ldots, \neg pesQ_{o+s}(\overrightarrow{y_{o+s}}), \overrightarrow{y_{i,k'}} \neq \overrightarrow{y_{m_{k'}+j,k'}}, \\ & i \in [1, m_{k'}],\ j \in [1, n_{k'}], k^{'} \in [1, t'], c. \end{aligned}$$

This clause is derived from the following general clause in Π:

$$\begin{aligned} Q_u(\overrightarrow{x}) \quad \leftarrow \quad & Q_1(\overrightarrow{y_1}), \ldots, Q_o(\overrightarrow{y_o}), \neg Q_{o+1}(\overrightarrow{y_{o+1}}), \ldots, \neg Q_{o+s}(\overrightarrow{y_{o+s}}), p_1(\overrightarrow{x_{1,1}}), \ldots, \\ & p_1(\overrightarrow{x_{m_1,1}}), \neg p_1(\overrightarrow{x_{m_1+1,1}}), \ldots, \neg p_1(\overrightarrow{x_{m_1+n_1,1}}), \ldots, p_k(\overrightarrow{x_{1,k}}), \ldots, \\ & p_k(\overrightarrow{x_{m_k,k}}), \neg p_k(\overrightarrow{x_{m_k+1,k}}), \ldots, \neg p_k(\overrightarrow{x_{m_k+n_k,k}}), \ldots, p_t(\overrightarrow{x_{1,t}}), \ldots, \\ & p_t(\overrightarrow{x_{m_t,t}}), \neg p_t(\overrightarrow{x_{m_t+1,t}}), \ldots, \neg p_t(\overrightarrow{x_{m_t+n_t,t}}), c. \end{aligned}$$

As evident from above, all literals used positively (first o literals) are translated to *optQ* in the body of $optQ_u(\overrightarrow{x})$ and the rest s, *i.e.*, those used negatively, are translated to $\neg pesQ$ in this clause's body. Since each *optQ* literal in the body of $optQ_u(\overrightarrow{x})$ must belong to $T^+_{P_S}\!\uparrow\!\alpha$ for it to belong to $T^+_{P_S}\!\uparrow\!\alpha+1$, using the inductive hypothesis, it can be shown that $c\ |Q_1(\overrightarrow{y_1})$ to $c\ |Q_o(\overrightarrow{y_o}) \in \uparrow\!\gamma$ for some γ. Similarly, the fact that $c\ |optQ_u(\overrightarrow{x}) \in T^+_{P_S}\!\uparrow\!\alpha+1$, it entails that constrained atoms corresponding to each *pesQ* literal in its body belong to $T^-_{P_S}\!\uparrow\!\alpha$. Consider any one such atom, say $c\ |pesQ_{o+j}(\overrightarrow{x}) \in T^-_{P_S}\!\uparrow\!\alpha$. As already shown, this entails that $c\ |Q_{o+j}(\overrightarrow{x}) \not\in T^+_P\!\uparrow\!\gamma$. Hence, for some definition of private predicates, $c\ |Q_{o+j}(\overrightarrow{x}) \in \uparrow\!\gamma$. Finally, because $\overrightarrow{y_{i,k'}} \neq \overrightarrow{y_{m_{k'}+j,k'}}$, $i \in [1, m_{k'}]$, $j \in [1, n_{k'}]$, $k' \in [1, t']$, for some definition of private predicates $p_{k'}(\overrightarrow{y_{i,k'}})\theta \in \uparrow\!\gamma$ and $p_{k'}(\overrightarrow{y_{m_{k'}+j,k'}})\theta \in \uparrow\!\gamma$. Hence, a Π_D can be constructed such that $Q_u(\overrightarrow{x})\theta \in \uparrow\!\gamma+1$. Similarly, the limit ordinal case can be shown in a straightforward manner.

($\Leftarrow$) We show that $c\ |Q_u(\overrightarrow{x}) \in \uparrow\!\omega$ entails $c\ |optQ_u(\overrightarrow{x}) \in T^+_{P_S} \uparrow \omega$ by reversing the arguments given above. We again use induction on the steps of construction. The inductive hypothesis states that $c\ |Q_u(\overrightarrow{x}) \in \uparrow\!\alpha$ entails $c\ |optQ_u(\overrightarrow{x}) \in T^+_{P_S}\!\uparrow\!\beta$ for some β.

For the base case $\uparrow 0 = \emptyset$, hence the hypothesis trivially holds. For successor ordinals, we assume that the hypothesis holds for all atoms added to in the step α. Next consider the step $\alpha+1$ and a $c\ |Q_u(\overrightarrow{x})$ literal added in this step, whose general form is as shown above. Because $c\ |Q_u(\overrightarrow{x}) \in \uparrow\alpha+1$, each literal used positively in its body, *i.e.*, the first o literals $c\ |optQ_1(\overrightarrow{y_1})$ to $c\ |optQ_o(\overrightarrow{y_o})$ can be shown to belong to $T^+_{P_S}\uparrow\gamma$, using the inductive hypothesis. Similarly, because the negatively used literals in $Q_u(\overrightarrow{x})$ must belong to $\uparrow\alpha$, they cannot be members of $T^+_P\uparrow\alpha$ and hence $c\ |pesQ_{o+j}(\overrightarrow{y_{o+j}}) \in T^-_{P_S}\uparrow\gamma$ for $j \in [1, s]$. Finally, all the sensitive literals used positively, (*i.e.*, $c\ |p_k(\overrightarrow{x_{m_k,k}})$) must belong to $\uparrow\alpha$ and each sensitive literal used negatively, (*i.e.*, $c\ |p_k(\overrightarrow{x_{m_k+j,k}})$) must belong to $\uparrow\alpha$ for $c\ |Q_u(\overrightarrow{x}) \in \uparrow\alpha+1$. Therefore, $X_{m_k+j,k} \neq X_{m_k,k}$ can now be shown. It is now straightforward to see that this entails $c\ |optQ_u(\overrightarrow{x}) \in T^+_{P_S}\uparrow\gamma+1$. Similarly, the limit ordinal case can be shown in a straightforward manner. $\square$

Relationship between Π_R, Π_{suf} and Π_{nec} We relate the satisfaction of sufficient policy and necessary policy to the satisfaction of original policy. That is, we wish to show that the transformation algorithm generates ‘correct’ necessary and sufficient policies. Informally, like the result above, the next results essentially state that whenever sufficient policy is satisfied, the original policy is also satisfied, irrespective of how the facts in Π_D are constructed. Similarly, satisfaction of necessary policy means that there is one such way to define the facts in Π_D such that the original policy will be satisfied. Hence, the relation between the Π_{suf} and Π_R and Π_{nec} and Π_R follows from the above theorem.

In each program P, P_S and P_N, a message is accepted if $c\ |accept(\overrightarrow{msg}) \in T_P^+\!\uparrow\!\omega$, $c\ |accept_{suf}(\overrightarrow{msg}) \in T_{P_S}^+\!\uparrow\!\omega$ and $c\ |accept_{nec}(\overrightarrow{msg}) \in T_{P_N}^+\!\uparrow\!\omega$ respectively. Hence the following corollaries hold.

Corollory 2. *Given a message M, a policy ruleset* Π_R*, and a set of facts* Π_D *defining private predicates (*p_k*,* $k \in [1, t]$*) that occur in* Π_R*,* $c\ |accept(\overrightarrow{m}) \in T_P^+ \uparrow \omega$ *if* $c\ |accept_{suf}(\overrightarrow{m}) \in T_{P_S}^+ \uparrow \omega$*.*

Proof: The $accept_{suf}()$ clause in Π_{suf} is defined as

$$accept_{suf}(\overrightarrow{m}) \leftarrow pesAllow(\overrightarrow{m}), \neg optDisallow(\overrightarrow{m})$$

It follows from theorem 4 that $c\ |pesAllow(\overrightarrow{m}) \in T_{P_S}^+ \uparrow \omega$ if $c\ |\ allow(\overrightarrow{m}) \in T_P^+\!\uparrow\!\omega$ for all definitions of private predicates. Also, $c\ |\ optDisallow(\overrightarrow{m}) \in T_{P_S}^- \uparrow \omega$ if and only if $c\ |\ disAllow(\overrightarrow{m}) \in T_P^- \uparrow\!\omega$ for some definition, therefore, $c\ |accept(\overrightarrow{m}) \in T_P^+\!\uparrow\!\omega$ if $c\ |accept_{suf}(\overrightarrow{m}) \in T_{P_S}^+ \uparrow \omega$ □

Corollory 3. *Given a message M, a policy ruleset* Π_R*, and a set of facts* Π_D *defining private predicates (*p_k*,* $k \in [1, t]$*) that occur in* Π_R*,* $c\ |accept_{suf}(\overrightarrow{m}) \in T_{P_S}^+ \uparrow \omega$ *if* $c\ |accept(\overrightarrow{m}) \in T_P^+ \uparrow \omega$ *for all possible definitions of predicates* p_k*,* $k \in [1, t]$*.*

Proof: Follows from theorem 4 and the definition of the predicate accept$_{suf}$ □

Corollory 4. *Given a message M, a policy ruleset* Π_R*, and a set of facts* Π_D *defining private predicates (*p_k*,* $k \in [1, t]$*) that occur in* Π_R*,* $c\ |accept_{nec}(\overrightarrow{m}) \in T_{P_N}^+ \uparrow \omega$ *if* $c\ |accept(\overrightarrow{m}) \in T_P^+ \uparrow \omega$*.*

Proof: The $accept_{nec}()$ clause in Π_{nec} is defined as

$$accept_{nec}(\overrightarrow{m}) \leftarrow optAllow(\overrightarrow{m}), \neg pesDisallow(\overrightarrow{m})$$

It follows from theorem 4 that $c \mid optAllow(\overrightarrow{m}) \in T^{+}_{P_N} \uparrow \omega$ if and only if $c \mid allow(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ for some definition of private predicates. Also, $c \mid pesDisallow(\overrightarrow{m}) \in T^{-}_{P_N} \uparrow \omega$ if and only if $c \mid allow(\overrightarrow{m}) \in T^{-}_{P} \uparrow \omega$ for all definitions of private predicates, therefore, $c \mid accept_{nec}(\overrightarrow{m}) \in T^{+}_{P_N} \uparrow \omega$ if $c \mid accept(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ □

Semantic proximity of Π_{suf} and Π_{nec} to the original policy In this section we wish to show that the transformed policies are semantically closest to the original policy. This is important because while it may always be possible to generate a 'safe' policy which is completely unrelated to a user's policy, our transformation algorithm generates 'safe' policies that are semantically closest to the original policy. Informally, we prove below that there is no policy that is safe and semantically closer to the original policy than our transformed policies.

To achieve the result explained above, we first define a dominance relation for policies with respect to acceptance of messages by a policy. While keeping a message fixed, the result of different policy applications on that message gives the following dominance relation:

Definition 42 (Email Policy Dominance Relation)**.** *Relation on $D_\Pi \times D_\Pi$ is said to be the email policy dominance relation, where D_Π is the domain of all policies, such that an ordered pair $\langle \Pi_X, \Pi_Y \rangle \in$ if and only if for all messages, M, $c|accept_X(\overrightarrow{m}) \in T^{+}_{P_X} \uparrow \omega$ whenever $c|accept_Y(\overrightarrow{m}) \in T^{+}_{P_Y} \uparrow \omega$, in which $P_X = \Pi_X \cup M$ and $P_Y = \Pi_Y \cup M$. We represent the fact $\langle \Pi_X, \Pi_Y \rangle \in$ by $\Pi_X \Pi_Y$.*

We say that a policy A entails a policy B whenever A $>_\Pi$ B.

Proposition 1. *Policy Dominance Relation on D_Π is a partial order.*

Proof: For all policies x, y and z, each of the following hold, and therefore the proof follows:

- xx
- $xy \wedge yx \rightarrow x = y$
- $xy \wedge yz \rightarrow xz$ $\square$

Proposition 2. *$\Pi_{suf} >_{\Pi} \Pi_R \cup \Pi_D$ where Π_{suf} is the sufficient transformation of Π_R*

Proof: Follows from Corollary 3 $\square$

Proposition 3. *$\Pi_R \cup \Pi_D >_{\Pi} \Pi_{nec}$ where Π_{nec} is the necessary transformation of Π_R*

Proof: Follows from Corollary 4 $\square$

Theorem 5. *Given a policy ruleset Π_R and a set of facts, Π_D defining private predicates p_k, k ∈ [1, t] that occur in Π_R and any message M, Π_{suf}, the sufficient transformation of Π_R, is the least upper bound, under the policy dominance relation , that entails $\Pi_R \cup \Pi_D$, for all possible definitions of p_k, k ∈ [1, t].*

Proof: From proposition 2 – Π_{suf} (Π_R∪ Π_D) and from corollary 3 – if c|accept$_{suf}(\overrightarrow{m})$ $\in T^{+}_{P_S} \uparrow \omega$ then c|accept$(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ for all possible definitions of p$_k$, k ∈ [1, t]. To show that Π_{suf} is the least such policy, we assume otherwise and give the proof by contradiction. Assume there is a policy Π_X such that $\Pi_{suf} >_{\Pi} \Pi_X >_{\Pi} \Pi_R \cup \Pi_D$ and c|$accept_X(\overrightarrow{m}) \in T^{+}_{P_X} \uparrow \omega$, where P$_X$ is the program $P_X \cup M$, entails c|accept$(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ for all possible definitions of private predicates (where P = $\Pi_R \cup \Pi_D \cup M$).

Consider a message M such that c|accept$_X(\overrightarrow{m}) \in T^+_{P_X} \uparrow \omega$, and c|accept$_{suf}(\overrightarrow{m})$ $\notin T^+_{P_S} \uparrow \omega$. Due to the entailment assumption above c|accept$(\overrightarrow{m}) \in T^+_P \uparrow \omega$ for all possible definitions of the private predicates. But from corollary 3 if c|accept$(\overrightarrow{m})$ $\in T^+_P \uparrow \omega$ for all possible definitions of private predicates then c|accept$_{suf}(\overrightarrow{m}) \in$ $T^+_{P_S} \uparrow \omega$, which contradicts the assumption. □

Theorem 6. *Given a policy ruleset* Π_R *and a set of facts,* Π_D *defining private predicates* p_k*,* $k \in$ *[1, t] that occur in* Π_R *and any message M,* Π_{nec}*, the necessary transformation of* Π_R*, is the greatest lower bound, under the policy dominance relation , that is entailed by* $\Pi_R \cup \Pi_D$*, for some definition of predicates* p_k*,* $k \in$ *[1, t].*

Proof sketch: The proof follows from theorem 4 and corollary 4 following similar arguments as in theorem 5 □

6.5.4 Protection achieved from transformed policies

Now that we have defined the relation of our transformed policies to the original policy, we are ready to describe how these transformed policies achieve desired security goal that is addressed in this chapter.

To describe the protection achieved from evaluating transformed policies instead of the original policy, we compare following three cases of attacker knowledge.

1. **Default:** The attacker can compute $\Pi^{\omega *}_R$ and verify if the constrained atoms c|accept$(\overrightarrow{m}) \in T^+_P \uparrow \omega$ or $T^-_P \uparrow \omega$ for different m.

2. **Knowledge of transformations:** The attacker is only allowed to know the transformed policy completions $\Pi^{\omega *}_{nec}$ and $\Pi^{\omega *}_{suf}$ and can generate the verifiers, *viz.*, c|accept$_{nec}(\overrightarrow{m})$ and c|accept$_{suf}(\overrightarrow{m})$ for different m.

3. **Original ruleset with Π_{nec}, Π_{suf} verifiers:** Attacker can compute $\Pi_R^{\omega *}$ but only generate the verifiers – c|accept$_{suf}(\overrightarrow{m})$, c|accept$_{nec}(\overrightarrow{m})$ for different m.

In each case, the attacker may know either the original ruleset or the transformed rules (Π_{nec} and Π_{suf}). Depending upon which policies are used to evaluate message acceptance, the corresponding verifiers are generated. Hence, in the default case the original policy is used to evaluate messages, whereas in the other two cases the transformed policies are evaluated. These are formally proved next.

Theorem 7. *It is possible to verify a guess of recipient's policy through guessing attacks if the attacker knows Π_R that includes leaky predicates and messages M_i, $i > 0$ are accepted by the program $\Pi_R \cup \Pi_D \cup M_i$*

Proof: Because evaluations are carried out using $\Pi_R \cup \Pi_D \cup M_i$, leaky predicates will make available verifiers c|accept$(\overrightarrow{m})$ to the attacker. Since the attacker can compute $\Pi_R^{\omega *}$, she can generate a sequence of messages to verify her guess. □

Theorem 8. *If the attacker knows $\Pi_{suf}^{\omega *}$ (resp. $\Pi_{nec}^{\omega *}$) and acceptance of messages is decided by evaluation of $\Pi_{suf} \cup \Pi_D \cup M_i$ (resp. $\Pi_{nec} \cup \Pi_D \cup M_i$), then it is not possible to verify that $g \in$ ext(p), where $p \in \mathcal{P}$ and ext(p) is its extension*

Proof: Because private predicates defined in Π_D do not occur in rulesets Π_{nec} and Π_{suf}, they don't occur in $\Pi_{suf}^{\omega *}$ and $\Pi_{nec}^{\omega *}$. Therefore, with the sets $\Pi_{suf}^{\omega *}$ and $\Pi_{nec}^{\omega *}$, the attacker doesn't have enough information to construct the definition of predicates in R_A from policy rulesets, and acceptance verifiers. □

Theorem 9. *If the attacker knows Π_R and acceptance of messages is decided by evaluation of $\Pi_{suf} \cup \Pi_D \cup M_i$ (resp. $\Pi_{nec} \cup \Pi_D \cup M_i$), then it is not possible to verify that $g \in$ ext(p), where $p \in \mathcal{P}$ and ext(p) is its extension*

Proof: With Π_R the attacker can construct rules that define a predicate $\mathrm{p}' \in R_A$ such that $\mathrm{ext}(\mathrm{p}') \subseteq \mathrm{ext}(\mathrm{p})$. However, these rules require verifiers generated from evaluation of the $\Pi_R \cup \Pi_D \cup M$. Because $\mathrm{p} \not\propto \Pi_{suf} \cup M$ (and $\mathrm{p} \not\propto \Pi_{nec} \cup M$), therefore, verifiers generated from the evaluation of this program is exactly the same for both the cases:

1. $\mathrm{g} \in \mathrm{ext}(\mathrm{p})$

2. $\mathrm{g} \notin \mathrm{ext}(\mathrm{p})$

Hence, the attacker does not get back the required verifiers to verify her guess. $\square$

6.6 Chapter conclusion

In this chapter we identified an undesirable side effect of combining different email-control mechanisms, namely, the leakage of sensitive information. Even though confidentiality of sensitive information has been widely studied as a research problem, it assumes a different form in the email context, because of the ease with which sensitive information is leaked. We provide example scenarios where leakage is made possible in two ways – using the message delivery protocol itself and using leakage channels beyond the mail delivery protocol. Based on how these leakages may be used by an attacker, we categorize them into two classes – automatic generation of acknowledgement receipts for validating an email address and automatic generation of acknowledgments for inferring private information about the recipient. As leakage channels beyond the control of the delivery protocol can't be closed by modifying email delivery protocol alone, preventing leakages is hard to achieve. In particular, we investigate in detail the second class of attacks where a victim's sensitive information is leaked.

We defined an attacker model for uncovering recipients' sensitive information. In the worst case scenario, we assume that the attacker knows recipient's mail acceptance criteria, but not the sensitive information maintained by the recipient. With the abilities of computing Clark completion of normal Horn clauses, unfold/fold transformations and generating messages, the attacker can mount attacks that leak sensitive information. As a solution, we provide an algorithmic transformation which can sanitize the combination of email-control mechanisms, so that the leakage is prevented. We also show that the transformed policies that we generate are 'closest' semantically to the original policy.

Here, we are not concerned with feedback obtained from out of band channels, like recipient informing the sender through a telephone conversation. What we do aim to provide is a guarantee that the system itself will not signal whether message acceptance depended upon private information maintained at the recipient's end. It is possible to construct a hierarchy of mechanisms to control email delivery [55], where our transformation can be supported through message evaluation at different levels. For example, if all recipients in an email domain use the same sensitive predicate, then that predicate can be pushed upstream, making it a global acceptance criteria, thereby reducing it's sensitivity. In our ongoing work, we are studying such techniques to enhance our methodology.

$$Q_u(\overrightarrow{Y_u})\colon -\ Q_1(\overrightarrow{Y_1}), \ldots, \neg Q_v(\overrightarrow{Y_v}), p_1(\overrightarrow{X_{1,1}}), \ldots, p_1(\overrightarrow{X_{m_1,1}}), \neg p_1(\overrightarrow{X_{m_1+1,1}}), \ldots,$$

$$\neg p_1(\overrightarrow{X_{m_1+n_1,1}}), \ldots, p_{t'}(\overrightarrow{X_{t',1}}), \ldots, \neg p_{t'}(\overrightarrow{X_{m_{t'}+n_{t'},t'}}), c.$$

For each clause in Π_R as shown above, add following clauses, for each k and u, if not already present: (with $i \in [1, m_{k'}]$, $j \in [1, n_{k'}]$ and $k' \in [1, t']$)

$$pesQ_u(\overrightarrow{Y_u})\colon -Q_u matchP_{t'}(\overrightarrow{X_{t'}}, \overrightarrow{m}), Q_u matchNotP_{t'}(\overrightarrow{X_{t'}}, \overrightarrow{m}).$$

$$Q_u matchP_k(\overrightarrow{X_{m_k+j}}, \overrightarrow{m})\colon -pesQ_1(\overrightarrow{Y_1}), \ldots, \neg optQ_v(\overrightarrow{Y_v}), Q_u matchP_1(\overrightarrow{X_{1,1}}, \overrightarrow{m}), \ldots,$$

$$Q_u matchP_1(\overrightarrow{X_{m_1,1}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{1,k}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{m_k,k}}, \overrightarrow{m}),$$

$$Q_u matchNotP_k(\overrightarrow{X_{m_k+1,k}}, \overrightarrow{m}), \ldots, Q_u matchNotP_k(\overrightarrow{X_{m_k+(j-1),k}}, \overrightarrow{m}),$$

$$Q_u matchNotP_k(\overrightarrow{X_{m_k+(j+1),k}}, \overrightarrow{m}), \ldots, Q_u matchNotP_k(\overrightarrow{X_{m_k+n_k,k}}, \overrightarrow{m}), \ldots,$$

$$Q_u matchP_{t'}(\overrightarrow{X_{1,t'}}, \overrightarrow{m}), \ldots, Q_u matchNotP_{t'}(\overrightarrow{X_{m_{t'}+n_{t'},t'}}, \overrightarrow{m}), \overrightarrow{X_{i,k'}} \neq \overrightarrow{X_{m_{k'}+j,k'}}, c.$$

$$Q_u matchNotP_k(\overrightarrow{X_i}, \overrightarrow{m})\colon -pesQ_1(\overrightarrow{Y_1}), \ldots, \neg optQ_v(\overrightarrow{Y_v}), Q_u matchP_1(\overrightarrow{X_{1,1}}, \overrightarrow{m}), \ldots,$$

$$Q_u matchP_1(\overrightarrow{X_{m_a,1}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{1,k}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{i-1,k}}, \overrightarrow{m}),$$

$$Q_u matchP_k(\overrightarrow{X_{i+1,k}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{m_k,k}}, \overrightarrow{m}), Q_u matchNotP_k(\overrightarrow{X_{m_k+1,k}}, \overrightarrow{m}), \ldots,$$

$$Q_u matchNotP_k(\overrightarrow{X_{m_k+n_k,k}}, \overrightarrow{m}), \ldots, Q_u matchP_{t'}(\overrightarrow{X_{1,t'}}, \overrightarrow{m}), \ldots,$$

$$Q_u matchNotP_{t'}(\overrightarrow{X_{m_{t'}+n_{t'},t'}}, \overrightarrow{m}), \overrightarrow{X_{i,k'}} \neq \overrightarrow{X_{m_{k'}+j,k'}}, c.$$

$$optQ_u(\overrightarrow{Y_u})\colon -optQ_1(\overrightarrow{Y_1}), \ldots, \neg pesQ_v(\overrightarrow{Y_v}), \overrightarrow{X_{i,k'}} \neq \overrightarrow{X_{m_{k'}+j,k'}}, c.$$

Figure 6.1: Transformation algorithm

Chapter 7: Extensions to SMTP

7.1 Introduction

In this chapter we exploit the SMTP extension model to incorporate additional service models for supporting policy-driven message evaluations and message negotiations.

7.2 Extensions to SMTP

In this section we present the extensions to the SMTP protocol required to support our scheme. The extensions we discuss incorporate our earlier proposal [42]. In addition to the earlier extensions, we need extensions to help us advertise policies upstream, to communicate an explicit criteria for message acceptance to the SESP or the sender. This obviates the need to transmit messages to the RESP that can never satisfy the transmitted policy. We assume a transaction model as described in RFC 2821 [1] and that additional reply codes, for instance, `253,555`, *etc.*, can be used to communicate new information, like sequence of exchanges to be used, message rejection, *etc.* In the transaction model, the server initiating an SMTP session is called the SMTP client and the second server is called the SMTP server. Accordingly, we refer to SESP by the former and RESP by the later name in this section.

7.2.1 Requirements

New requirements to support policy and feedback communication upstream are summarized below. Our extensions are designed to address these new requirements:

- Support for communicating a policies and feedback for rejected message upstream on a per message basis. With this feature, recipient policies, *i.e.*, MRAP, can be advertised upstream, along with the MSP policy.

- Additional reply codes to denote temporary and permanent rejection of messages. Temporary reject means message refinement may reverse the reject decision.

- Support for a graded scheme of policy communication and feedback control, using the Service level agreement policy (SLAP).

The provision for upstream communications during SMTP handshake in [42] is inadequate for the purpose of providing policies and feedback on a per message basis. Clearly, at the time of the handshake, it cannot be known which recipients' policies to transmit. Abilities, characteristics, and interests of the diverse types of RESP's and SESP's, as well as the trust between the communicators, may dictate the time frequency, and the manner in which policy content is communicated by the RESP to the SESP. Next we list the various schemes that can be employed by the disparate mail servers on the internet:

T1: No policy or feedback communication.

T2: MSP policy advertisement and feedback, once per SMTP session.

T3: MSP policy advertisement and feedback, on per session and per message basis.

T4: MSP, MRAP advertisement and feedback, where MSP and its feedback is shared either on a per message basis, or once per session.

T5: MRAP advertisement and feedback.

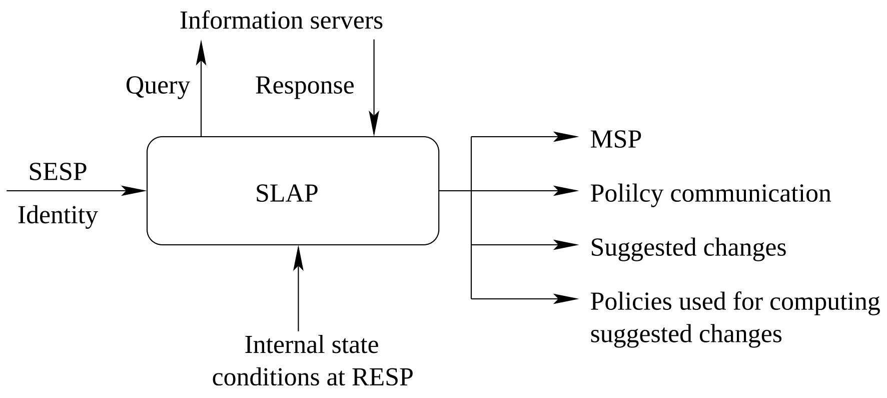

Figure 7.1: Service Level Agreement Policy (SLAP)

The choice of selecting between one of the above transmission schemes is entrusted to SLAP. In other words, SLAP must first evaluate the trustworthiness of a sender and its SESP and accordingly grant them the desired service level, as pictured in figure 7.1. In summary, it not only selects the MSP policy for the session, but also decides which policies to communicate, if any, and whether or not to provide suggestions for refinement. For example, if the SESP is a known spammer, possibly blacklisted, the SLAP can overrule any feedback in the form of policy advertisement or suggested changes to rejected messages (*i.e.*, T1). On the other extreme, when communicating with a trusted SESP, SLAP may be most forthright in providing useful feedback for message refinement (*i.e.*, T4). In view of these additional controls entrusted to it, the original SLAP characterization [42] has been extended here.

7.2.2 Extensions

As proposed earlier [42], we invoke a new handshake command, namely `SHLO`, to indicate the capability to support policy extensions to the SMTP protocol. One of the parameters of `SHLO` message is the maximum number of attributes that can be refined at a time. In the earlier work [42], presence or absence of MSP policy in the extended handshake served as an indication to the client regarding which transmission scheme was to be followed. With five possible alternatives possible in our approach, the simplistic approach taken earlier needs to be extended. We use new reply codes, described in table 7.1, for each transmission scheme for this purpose.

Table 7.1: Proposed additions to SMTP reply codes

SMTP phase	**Code**	**Parameters/Significance**
Handshake	253	None/No communication
	254	MSP, once per session
	255	MSP, per session and per message
	256	MSP, per session and per message MSP; Per message MRAP
	257	None, per message MRAP
Transmission	258	MSP or MRAP/ Per message
	259	None/ Response to challenge found incorrect
	555	σ_a/ Temporary reject
	556	None/ Permanent reject

New reply codes are introduced for use during the handshake, *i.e.*, `253--257`, and can carry policy content as a parameter. Similarly, the transmission phase also is extended with new reply codes, *viz.*, `258`– to communicate policy content; `259`– to indicate to the transmitter that its response to the supplied challenge was found incorrect; `555`– to reject a message with an answer constraint; and `556`– a permanent rejection code. In addition to the reply codes, the transmission phase requires two additional commands to be introduced. First command is needed for the SMTP client to respond to policy challenges, like *turing tests* [4], *etc.*, and is named `POLICY RESPONSE`. The second command indicates the readiness of the SMTP client to transmit message headers. We name this command `ALL HEADERS`, and it contains as a parameter, all the headers in the message being transmitted. These extensions will be clear when we discuss the extended state machine in the next section.

7.2.3 Extended SMTP state machine

Extensions to reply codes and commands, introduced above, are used to extend the SMTP client and server state machines. Next, we discuss the extensions to client state machine in detail, and corresponding extensions to the server are implied. In the figures shown below, each state (except those labeled R1, R2, R3, R4, R5), represented by a box, implies that the client has issued a command, the label of the box, and is waiting for the server's response. Also, the figures picture faithful runs of protocol, and therefore, we have dropped '503 Bad Sequence' state transitions. Extension to the SMTP handshake are discussed first.

Figure 7.2 pictures extensions to the client state machine for the extended SMTP handshake phase. The client starts in the 'Start' state with a non-empty message queue. It attempts connection to the SMTP server. A policy-enabled server replies with a `SHLO`. To this message the client issues a `SHLO` along with other parameters, like identity certificates, *etc.*, and enters the 'SHLO' state. The server's reply consists of a code from the set– {`253,254, 255,256,257`} that indicates the transmission scheme

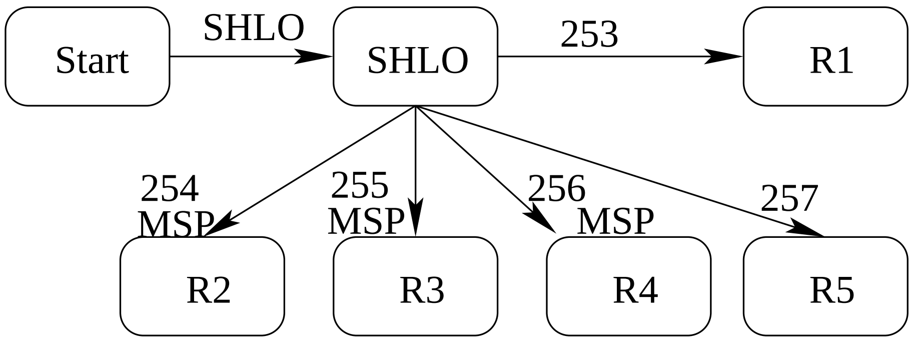

Figure 7.2: Modified extended handshake

to follow and a parameter, like the MSP policy or the number of refinable headers, if required. If MSP is communicated upstream, the SMTP client is expected to screen out or refine messages in the transmission queue based on the transmitted policy, and then change its state to denote the end of handshake phase. The corresponding states at the client are denoted by ‘R1’, ‘R2’, ‘R3’, ‘R4’, and ‘R5’ and are discussed next.

The state ‘R1’ represents the beginning of ‘no feedback, no policy advertisement’ transmission scheme, *i.e.*, T1. Essentially, this transmission scheme caters to clients that cannot support policy-based evaluation, in other words, the transmission scheme is identical to the current protocol. T1 could be used in cases where clients are policy-enabled, but are not trusted by the server. However, policies can still be evaluated at the server. If messages can’t satisfy the policies at server-end, this information is not leaked to the client. The server simply drops rejected messages silently.

The state ‘R2’ represents the beginning of T2 transmission scheme, *i.e.*, MSP policy communication, once per session, and MSP feedback. An additional state is required to verify portable MSP application in this machine. This state is the ‘All Headers’ state that denotes transmission of all message attributes prior to the

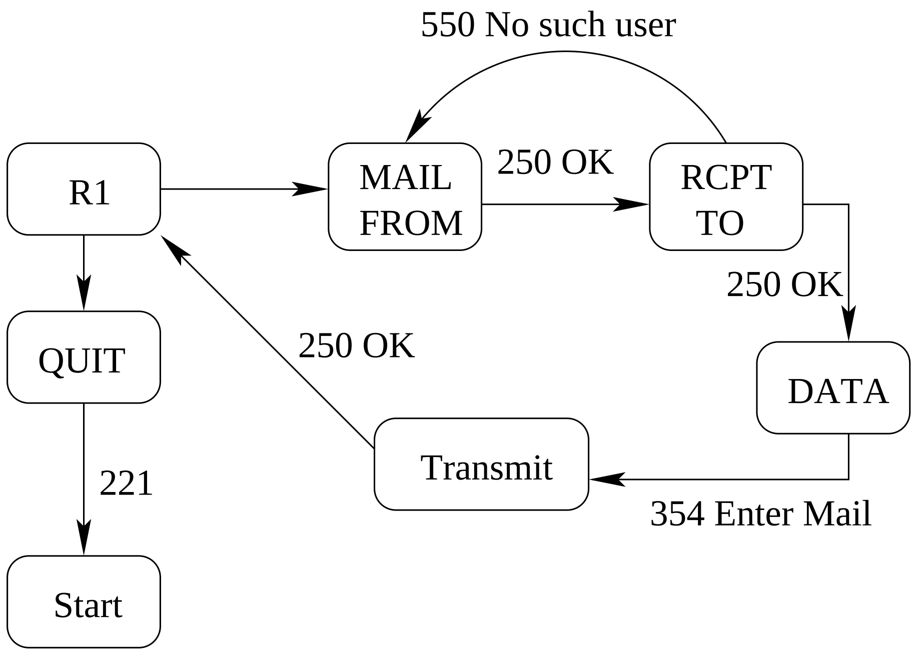

Figure 7.3: Transmission with no feedback.

transmission of the content. The server thus has an opportunity to verify if its MSP policy was faithfully applied, and reject the message otherwise. We also permit the use of '555 Denied' reply, to allow for communicating suggestions to refine rejected messages, in the case of local MSP evaluation. The state machine is pictured in figure 7.4.

The state 'R3' represents beginning of T3 transmissions. In this case, MSP could be used on a per message basis as well. For example, as a response to `MAIL FROM` command, the SMTP server (*i.e.*, RESP) can reply with a `258` message instead of the usual `250 OK`. The new reply code can carry an MSP policy, say, a memory bound function challenge whose correct response is a proof of computational effort [26]. Hence, the figure 7.5, shows an additional state 'Policy Response' where the client has sent a response to the MSP challenge and is waiting for its verification by the

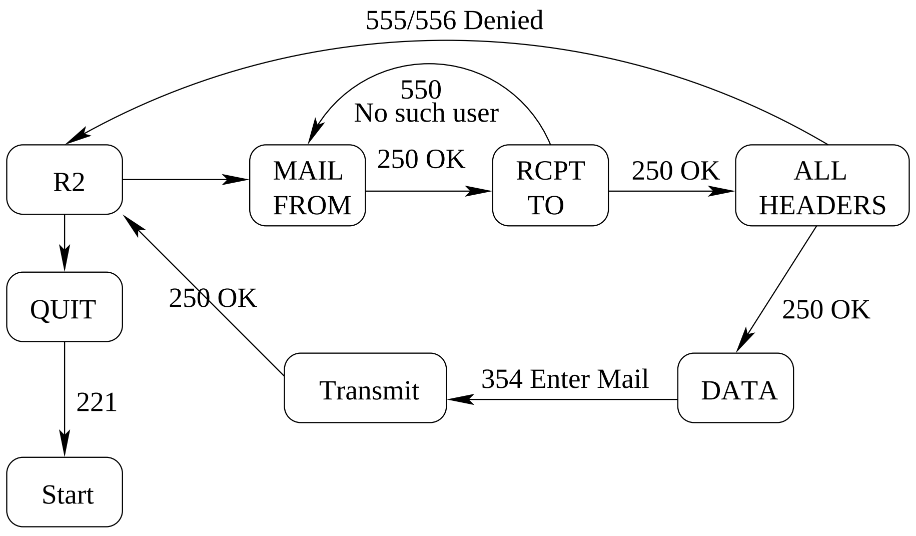

Figure 7.4: Transmission with MSP communicated on a per session basis.

server. The figure also shows that per message MSP scheme could be selectively applied, thus sparing senders who are trusted by the server.

T4 transmissions, pictured in figure 7.6, begin in the state 'R4'. R4 signifies per-session MSP, per-message MSP and the message recipient's MRAP based policy communication and feedback. Compared to the state machine involving R3, R4 needs two 'Policy Response' states, one for per message MSP and one for MRAP application. A recipient's MRAP can be communicated to the SMTP client when a message addressed to the recipient is being transmitted. Sharing of MRAP can cause the following: a) termination of transmission, because message or its refinements do not satisfy MRAP; b) change of state to 'All Headers' since the message either satisfies the MRAP or has been refined to meet its criteria; c) change of state to 'Policy Response' where MRAP challenge has been replied to. Also, the SMTP server can verify if its per session MSP was faithfully applied, and prepare an appropriate response to the client's 'All Headers' state. Similarly, the server can also verify application of MRAP

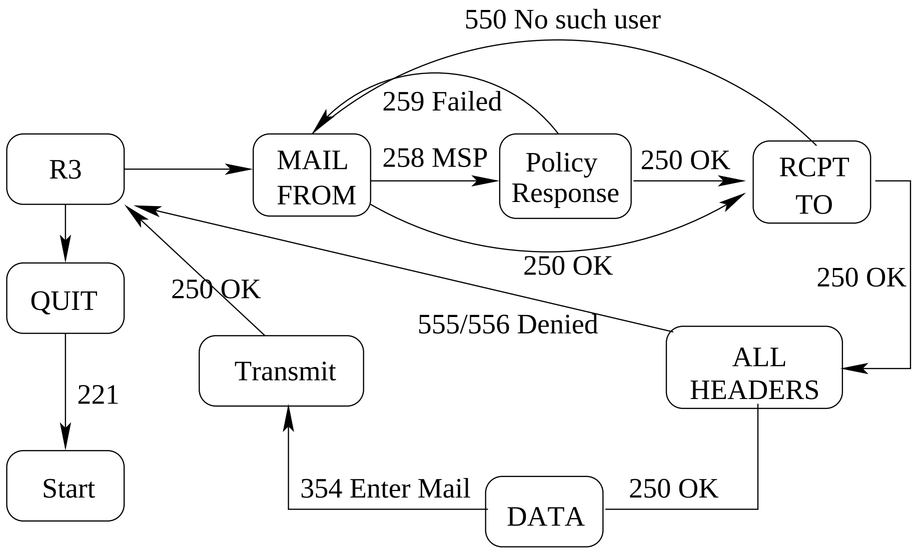

Figure 7.5: Transmission with MSP shared per session as well as on a per message basis.

and respond with a reject decision (with a possibility of communicating suggestions to refine it) to the client's 'Transmit' state when a message is found unacceptable.

Finally, T5 scheme is represented by the state machine beginning with 'R5' and is pictured in figure 7.7. R5 represents the state, in which the server decides to use only the individual MRAP policies. Compared to state machine with R4, this one has two less states because verification of session-based MSP and message-based MSP is not required. All other state transitions are the same as those defined for R4 state machine.

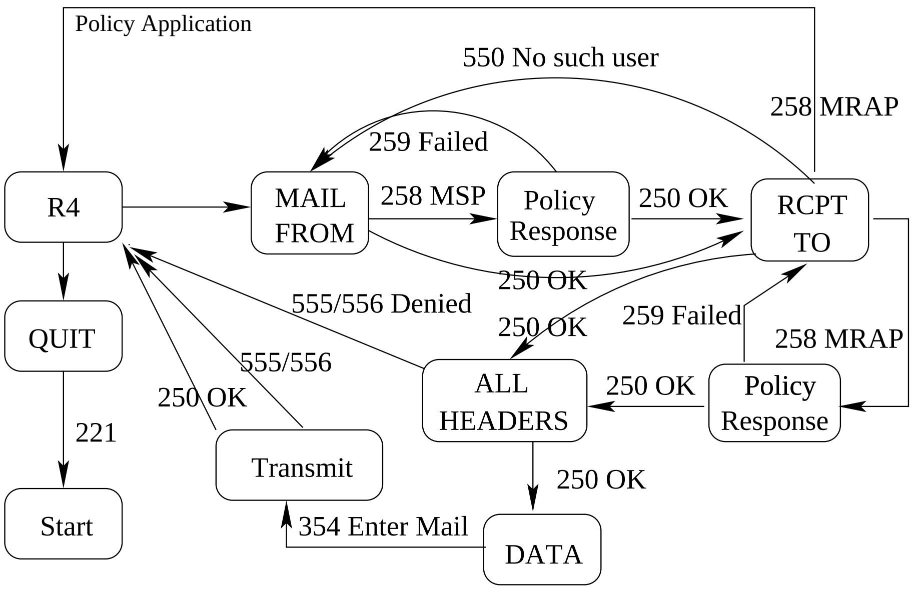

Figure 7.6: Transmission with MSP shared per session and per message and MRAP shared on a per message basis.

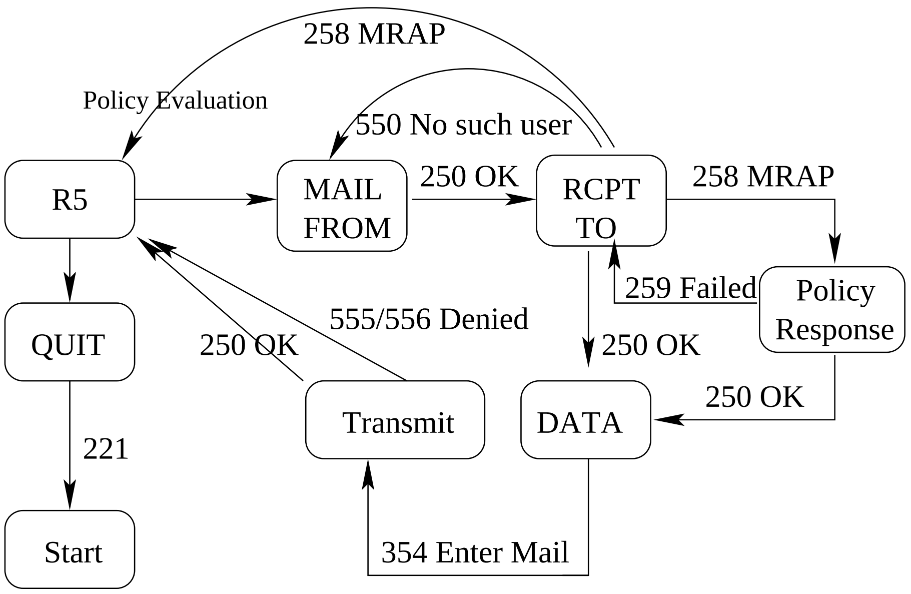

Figure 7.7: Transmission where only MRAP is shared on a per message basis.

Chapter 8: Securing Email by Design – A Web Services-based Case Study

8.1 Introduction

In addition to fixing conventional email systems, several researchers have called for secure replacements of email that satisfy current needs of the community. One such line of works is WSEmail [63] that argues replacement of existing email systems with a secure, web-services driven, email transmission network. The authors make their case with support for additional functional requirements, like *instant messaging*, in addition to protecting against several Misuse Cases.

In our view, a Web Services based email delivery service is an excellent choice for a replacement system. This is partly because of the ease with which distributed objects are accessed, allowing distributed support for solving the message acceptance problem. In other words, such an approach fits well with our aim of correlating various sources of information about incoming messages for making an effective decision regarding its legitimacy. Secondly, it presents us with an opportunity to express functional requirements in portable format that has a well-understood semantics. While requirements can be expressed by technologically unsophisticated users, the abstract descriptions can be shared, advertised or used in a machine independent manner.

In this chapter we explore the Web Services driven email transmissions and construct a secure, end-to-end design that is compared against functional and non-functional requirements of SMTP networks. Given that conventional email and our extensions can be used with mal-intent, we specify a collection of Misuse Cases that are shown to be thwarted by our design. Example standard Use Cases and Misuse Cases include the standard best-effort asynchronous delivery, preventing SPAM etc.; we extend to giving the recipient more control over message acceptance without allowing the misuse of inferring exact acceptance criteria. Our system, called WebMail, is a collection of web services that are orchestrated using the Business Process Execution Language (BPEL) [64].

In a similar effort, WSEmail [63] provides flexible means to communication, such as, dynamically discovering and negotiating communication protocols such as in Instant Messaging (IM), etc. AMPol [65] extends WSEmail by separating policies from delivery mechanisms, thereby achieving flexibility of operation. Our feedback-based recipient controlled email framework (Chapter 4) extended traditional SMTP-based email flows, thereby alleviating some annoying misuse cases of earlier systems. In this work, we collect best of the two approaches, by designing a comprehensive web-based solution using standard methods.

8.2 Use cases and Misuse cases

Next we relist the Use Cases and Misuse Cases for conventional SMTP systems. Our aim here is to be able to support most functional requirements for adoptability and ease of interfacing with existing systems, while preventing the Misuse Cases allowed by the conventional networks. Known Use Cases enabled by conventional SMTP-based mail are as follows:

Use case 1 *Best effort transmission of a text message from a sender (the principal actor) to a recipient (the secondary actor) through intermediate mail servers (auxiliary actors).*

Use Case 2 *Error reporting on transmission failure.*

A message transmission - broken down into three logical steps, is considered complete only if the message is routed to the recipient's mailbox. The steps are: from the sender to its email service providers (SESP); from SESP to recipients' email service provider (RESP); and finally from RESP to the recipient's mail box. However, physically, multiple mail servers may be involved and are subsumed under the logical entities - SESP and RESP. Message transmission may not be complete due to many failures, upon which the first point of failure detection is expected to inform the sender using another email message.

Specialization of Use Cases Above two Use Cases can be specialized for a variety of message types and properties of transmission channels. Standard use cases supported by SMTP implementations are given below.

1. Best-effort transmission of enhanced content including text and MIME messages [14].

2. Enforcing security mechanisms such as transmitting authenticated message and using encrypted channel.

3. Best-effort transmission of message acknowledgement receipts.

Best-effort relates to asynchronous transfer of messages across hosts on the internet, because recipient process may not be active when the sender process contacts them. In addition, SMTP extensions [20, 19] include commands and replies for source authentication and negotiations for establishing a secure channel for synchronous transmission. Finally, SMTP also facilitates delivery receipts in the form of another email. These Use Cases are supported by functionality built into communicating server processes.

Email delivery is subject to many misuse cases. These include lack of authentication, loss of privacy and integrity of content, vulnerability to unsolicited commercial email (spam), email bombs [9], *etc.*, of which our design prevents the following:

1. *Violating integrity and leaking or altering content*: Allowing unintended mal-actors to read message contents and alteration.

2. *Impersonating senders*: Allowing mal-actors to assume the identity of another person in a mail message.

3. *Email bombs*: This is a variation of DoS attack on email networks, where mail servers receive large number of messages, leading to denial of email service.

4. *Receiving undesirable email (spam)*: Allowing undesirable email to reach recipients' mailbox.

Although the STARTTLS [20] command exists in SMTP, most email messages are sent in clear-text over the wire, and stored as such at mail servers, thereby permitting the first misuse case to occur. Similarly, although the SMTP AUTH [19] exists, it requires prior exchange of secrets, which does not happen in most cases - thereby being subjected to the second misuse through sender-address spoofing. Current SMTP-like server designs result in being subjected to the third misuse case, where lack of recipient control over message delivery results in the last misuse case. Recent attention to spam has resulted in some proposals to add automated recipient controls to the message flow pipeline including, providing feedback about rejected messages [66]. A drawback of delivery controls is the inadvertent disclosure of acceptance criteria, that can now be used to defeat its purpose [67]. We also add this misuse case to the list of standard misuse cases and propose a solution in later in this chapter.

8.3 Web Services for message transmission

In this section we begin with the basic technical details of our model. Three basic components are considered for our specifications. First, we describe the types and parts of messages that are exchanged between Web Services. Then, we specify various Web Services that constitute the WebMail family. Finally, we specify various orchestrations of the WebMail family using BPEL process specifications.

Recall figure 3.1 (principals and policies in an email pipe) that details actors involved in message delivery. There is a direct translation of each principal there to a set of Web Services. That is, in a Web Services based message transmission, we replace each actor by one or more Web Services. Together these Web Services form a family, referred to as the *WebMail family*. Different *orchestrations* of these Web Services provide many *flavors* of email transmissions. Here, we show how to achieve these flavors. In effect, we show that earlier solutions for conventional systems can be readily adapted for the Web Services environment and possibly improved upon.

8.3.1 Message types

Table 8.1: Basic types of message elements

Type Name	**Primitive Type**	**Example**
MIME	ASCII string	Application/PDF
PKISignature	ASCII String	463hfd47654
Message ID	Long Int	239809832092
MType	Character string	Urgent, Personal, ...
WantAck	Boolean	Yes/No
Number	Positive Int	100
Nonce	Positive Int	10000
Email Address	ASCII string	abc@xyz.com
Password	ASCII string	******
Answer	ASCII string	Xy3

Message types define the protocol used for communication, *i.e.*, service interfaces are understood in terms of their input and output message. Here, we limit the types of transported objects, however, our list is extensible and it is possible to include the complete set of MIME[8] objects. Basic types are described in table 8.1, and complex (i.e. structural) types are described in table 8.2. We give these type definitions for completion. We don't intend to leverage on their type structure in this chapter. Our code (shown later) can be modified to be used with other typed structures as well. For instance, several techniques use custom structures for 'time' or 'credential', *etc.*, so we simply refer to them using an XML namespace element. Please note that we use the characters '*', '?', and '+' in the same sense of use as in BPEL manual [64], *i.e.*, .'*' means zero or more repetitions, '?' meaning zero or one occurrence and '+' means one or more repetitions.

Elements described in table 1 are expressed in Web Services Description Language (WSDL) [7] in the syntax shown in figure 8.1 (we omit all the details here).

```
<types>

    <schema targetNS="uri1" xmlns="schema1">

        <element name="Content" type="String">

        </element>

        .

        .

        <element name="Turing Test">

            <complexType>

                <all>

                    <element name="Image" type="Application/JPEG"/>

                    <element name="Answer" type="String"/>

                </all>

            </complexType>

        </element>

    </schema>

</types>
```

Figure 8.1: Basic Types in WSDL

Table 8.2: Complex types of message elements

Type Name	Type Structure	Example
Time	XmlNS=URI#Time	10:00 A.M EST
Key Pair	Int $\times$ Int	(53,97)
Credential	XmlNS=URI#Cred	Credential struct
Image	XmlNS=URI#Jpeg	JPEG struct
AObject	Application/Type	PDF file
Credential Chain	Credential*	Cred1,. . ., CredN
Currency	Enum: \$, £	\$, £
Bond	XmlNS=URI#Bond	\$3.5 Cred 1
Turing Test	Image	10101. . .01,
Turing Test Reply	Image $\times$ Answer	(10101. . .01, xy3)
Content	String?, AObject*	"Example", Image

8.3.2 Messages

Message types (summarized in table 5) are described next. Structure of a mail message is presented first. This message contains routing information, objects to be transmitted and additional attributes that aid the delivery of the message. Message attributes are used by downstream processes to make routing decisions [66]. Mail messages are described in WSDL format in figure 8.2 . In the following listings character '*' signifies zero or more repetitions, '?' zero or one occurrence and '+' means one or more repetitions.

In addition to mail messages, clients and servers transmit several other types of message enable underlying communication protocols by informing the status of the communication, properties of the transmission (QoS,) *etc.* (listed in table 8.3). Their WSDL syntax is shown in figures 8.3, 8.4 and 8.5.

Message definitions in table 3 determine the application data or the payload for the message communications. Table 4 defines protocol data exchanged for effectively

```
<message name="MailMessage">

    <part name="From" element="EmailAddress"/>+

    <part name="To" element="EmailAddress"/>+

    <part name="Date" element="Time"/>+

    <part name="ID" element="Message ID"/>+

    <part name="Surety" element="Bond"/>?

    <part name="Pass" element="Password"/>*

    <part name="Ack" element="WantAck"/>*

    <part name="Sign" element="PKISignature"/>*

    <part name="RTT reply" element="TuringTestReply"/>*

    <part name="MType" element="String"/>?

    <part name="Subject" element="String"/>?

    <part name="Body" element="Content"/>?

</message>
```

Figure 8.2: WSDL mail message

```
<message name="ReceiptNotice">

    <part name="Date" element="Time"/>+

    <part name="ID" element="MessageID"/>+

    <part name="Sign" element="PKISignature"/>*

</message>

<message name="FailNotice">

    <part name="Date" element="Time"/>+

    <part name="ID" element="MessageID"/>+

    <part name="Error" element="String"/>+

    <part name="Sign" element="PKISignature"/>*

</message>

<message name="RejectNotice">

    <part name="Date" element="Time"/>+

    <part name="ID" element="MessageID"/>+

    <part name="Eval Policy" element="Policy"/>+

    <part name="Sign" element="PKISignature"/>*

</message>

<message name="RefinementMessage">

    <part name="Date" element="Time"/>+

    <part name="ID" element="MessageID"/>+

    <part name="Sign" element="PKISignature"/>*

    <part name="Surety" element="Bond"/>*

    <part name="MType" element="String"/>?

    <part name="RTT" element="TuringTest"/>*

    <part name="Body" element="Content"/>*

</message>
```

Figure 8.3: WSDL Application Data

Table 8.3: Types of messages

Message type	**Utility**
Mail Message	Message to be delivered
Receipt notice	Notice of receipt and acceptance for delivery of a mail message
FailNotice	Notice of delivery failure
RejectNotice	Notice of delivery rejection
RefinementMessage	Changes desired in a mail message
RefinementFailure	Desired changes not possible
InformationMessage	Third party message evaluations
MailIntent	Indication of transmission intent
Service Level Accord	QoS for invocations
AcceptancePolicy	Acceptance rules advertisement
PKICertificate	Proof of identity and data secrecy

completing the task at hand. In particular, *Mail Intent*, message expresses the intent to send messages, *SLA* message is a response to mail intent message indicating number of messages allowed; while *Acceptance Policy* message states acceptable message attributes.

8.3.3 WSEmail family of Web Services

Next, we design a family of Web Services that perform various tasks to aid delivery of email messages. We list the set of externally callable methods for each principal involved in message delivery.

Sender's ESP (SESP) Sender's email service provider is designed to receive messages, route them to the destination, examine and *repair messages* before sending them, refine messages rejected by the RESP, *etc.*

1. SESPConnectPT

```
<message name="RefinementFailure">
    <part name="ID" element="MessageID"/>+
    <part name="RError" element="String"/>+
</message>
<message name="InformationMessage">
    <part name="Date" element="Time"/>+
    <part name="ID" element="MessageID"/>+
    <part name="Information" element="String"/>+
    <part name="Sign" element="PKISignature"/>*
</message>
```

Figure 8.4: WSDL Application Data (contd.)

2. SESPReceiveMsgPT
3. SESPAuthPT
4. SESPDeliveryPT
5. SESPMsgCallbackPT
6. SESPImprovementPT
7. SESPVirusExaminationPT
8. SESPVirusRemovalPT

Sender Sender's may need to expose a callback interface to receive rejection notices or notices for improving messages

1. SenderMsgCallbackPT
2. SenderMsgRefinementPT
3. SenderPasswdCallbackPT

```
<message name="MailIntent">

    <part name="Date" element="Time"/>+

    <part name="NoOfMsgs" element="Number"/>+

    <part name="Sign" element="PKISignature"/>*

</message>

<message name="ServiceLevelAgreement ">

    <part name="Date" element="Time"/>+

    <part name="AllowedNo" element="Number"/>+

    <part name="Sign" element="PKISignature"/>*

</message>

<message name="AcceptancePolicy">

    <part name="Date" element="Time"/>*

    <part name="Surety" element="Bond"/>*

    <part name="Sign" element="PKISignature"/>*

    <part name="RTTreply" element="TuringTest"/>*

    <part name="MType" element="String"/>*

    <part name="Body" element="Content"/>*

    <part name="Sign" element="PKISignature"/>*

</message>

<message name="PKICertificate ">

    <part name="Key" element="Credential"/>+

    <part name="Session" element="Nonce"/>*

</message>
```

Figure 8.5: WSDL Control Data

Recipient's ESP (RESP) Recipient's ESP provides the following set of services.

1. RESPHeloPT
2. RESP-TLSPT
3. RESPReceiveMsgPT
4. RESPVirusScanPT
5. RESPFilterPT
6. RESPControlPT
7. RESPSanitizationPT
8. RESPDeliveryPT
9. RESPStoragePT
10. RESPImprovementPT

Recipient A recipient need not expose any service; however, some recipients may allow their service providers to "push" messages to the recipient's host through the following service

1. RReceiveMsgPT

In addition to services provided by the SESP and RESP, third party services may be invoked during message transmission to check their desirability. Here we restrict ourselves to two Web Services, though this list could easily be extended.

Third party services RESP may invoke a distributed checksum service to verify if a message is a bulk message. Similarly, calls to escrow service to determine the validity of attached bonds is also possible.

1. CheckSumPT
2. BondVerificationPT

WSDL definitions of the Web Services, described above, are presented in figures 8.6, 8.7, 8.8, 8.9, 8.10, 8.11 and 8.12.

8.4 BPEL orchestration of WebMail

In this section, we begin with a basic set of synchronized Web Service invocations for message delivery. We present their interfaces - synchronous or asynchronous - for communication with other distributed processes. We illustrate typical activities, in BPEL notation by Andrews, Curbera [64], *et al.* SESP is described in figure 2 and RESP in figure 3, followed by their process descriptions (resp. listings 8 and 9). In the BPEL process specifications (figures 8.13 and 8.17) of processes we assume that $< partnerLink >$ elements, identifying the roles of involved services, are already specified. We omit namespace elements, variable declarations, *etc.*, and depend on the context for clarity.

8.4.1 SESP process specification

SESP process waits for messages from senders. Senders invoke SESP's ReceiveMsgPT. Once the message is received, two concurrent threads of execution begin, viz., scanning the received message's body for viruses and a UDDI query to locate the recipient's email service provider (RESP). If a message is found to be infected, the virus removal process is run after the scan is completed. Next, the SESP invokes the HeloPT service of the recipient to begin message delivery. Assuming that the RESP allows SESP to

```
<portType name="SESPReceiveMsgPT">

    <operation name="GetMessage">

        <input message="Mail Message"/>

        <output message="Receipt Notice"/>

        <fault name="Fail" message="FailNotice"/>?

    </operation>

</portType>


<portType name="SESPConnectPT">

    <operation name="GetSLA">

        <input message="MailMessage"/>

        <output message=" IntentMessage"/>

        <fault name="Fail" message="SLAFail"/>?

    </operation>

 </portType>


<portType name="SESPAuthPT">

    <operation name="AUTH">

        <output message="PKICertificate"/>

    </operation>

</portType>


</portType> <portType name="SESPCallbackPT">

    <operation name="MessageCallBack">

        <input message="RefinementMessage"/>

    </operation>

</portType>
```

Figure 8.6: WSDL portType specs for SESP services

```
<portType name="SESPDeliveryPT">

    <operation name="SendMessage">

        <input message="MailMessage"/>

        <output message="ReceiptNotice"/>

        <fault name="Fail" message="FailNotice"/>?

    </operation>


<portType name="SESPImprovementPT">

    <operation name="Refinement">

        <input message="RefinementMessage"/>

        <output message="MailMessage"/>

        <fault name="Fail" message="FailNotice"/>?

    </operation>

</portType>


<portType name="SESPExaminationPT">

    <operation name="VirusScan">

        <input message="Mail Message"/>

        <output message="Information Message"/>

    </operation>

</portType>
```

Figure 8.7: WSDL portType specs for SESP services(contd.)

```
<portType name="SESPVirusRemovalPT">

    <operation name="VirusRemoval">

        <input message="Mail Message"/>

        <output message="Mail Message"/>

    </operation>

</portType>
```

Figure 8.8: WSDL portType specs for SESP services(contd.)

transmit messages through a service level agreement (SLA), SESP invokes RESP's message receiving operation RecieveMsgPT.

Figures 8.13, 8.14, 8.15, 8.16 show a typical SESP process in BPEL syntax. The code has four main blocks: headers and type declarations (figure 8.13), message reception (figure 8.14); message preparation (figure 8.15); and message delivery (figure 8.16). The first part accepts messages from a sender, to be delivered to some recipient. In addition, the SESP process allows its message callback service to retransmit an earlier rejected (but now revised) message. In other words, messages rejected earlier, say for lack of authentication or other attributes desired by RESP, are repaired with the help of this feedback loop. Next, each message enqueued for delivery is subject to checks (like virus scan, *etc.*) to ensure good quality of each message. Finally, the message is sent across to the RESP.

8.4.2 RESP process specification

Next, we define an RESP process that enforces a sample service level agreement (SLA) and a reasonable message acceptance policy (AP), given (informal description) in table 8.4.

```
<portType name="RESPHeloPT">

    <operation name="SLAevaluation">

        <input message=" MailIntent"/>

        <output message="ServiceLevelAccord"/>

    </operation>

</portType>


<portType name="RESP-TLSPT">

    <operation name="STARTTLS">

        <input message="PKICertificate"/>

        <output message="PKICertificate"/>

    </operation>

</portType>


<portType name="RESPReceiveMsgPT">

    <operation name="GetMessage">

        <input message="MailMessage"/>

        <output message="ReceiptNotice"/>

        <fault name="Fail" message="FailNotice"/>?

        <fault name="Reject" message="RejectNotice"/>?

    </operation>

</portType>
```

Figure 8.9: WSDL portType specs for RESP services

```
<portType name="RESPVirusScanPT">
    <operation name="VirusScan">
        <input message="MailMessage"/>
        <output message="InformationMessage"/>
    </operation>
</portType>

<portType name="RESPFilterPT">
    <operation name="BayesianFiltering">
        <input message="MailMessage"/>
        <output message="InformationMessage"/>
    </operation>
</portType>

<portType name="RESPControlPT">
    <operation name="SenderRep">
        <input message="Sender"/>
        <output message="InformationMessage"/>
    </operation>
</portType>

<portType name="RESPSanitizationPT">
    <operation name="Sanitization">
        <input message="MailMessage"/>
        <output message="MailMessage"/>
    </operation>
</portType>
```

Figure 8.10: WSDL portType specs for RESP services(contd)

```
<portType name="RESPDeliveryPT">
    <operation name="SendMessage">
        <input message="MailMessage"/>
        <output message="ReceiptNotice"/>
        <fault name="Fail" message="FailNotice"/>?
   </operation>
</portType>

<portType name="RESPStoragePT">
    <operation name="StoreMessage">
        <input message="MailMessage"/>
        <fault name="Fail" message="FailNotice"/>?
    </operation>
</portType>

<portType name="RESPImprovementPT">
    <operation name="RefineMsg">
        <input message="MailMessage"/>
        <output message="RefinementMsg"/>
        <fault name="Fail" message="FailNotice"/>?
        <fault name="Reject" message="RejectNotice"/>?
    </operation>
</portType>
```

Figure 8.11: WSDL portType specs for RESP services(contd)

```
<portType name="CheckSumPT">

    <operation name="DCC">

        <input message="MailMessage"/>

        <output message="InformationMessage"/>

    </operation>

</portType>


<portType name="bondVerificationPT">

    <operation name="VerifyBond">

        <input message="MailMessage"/>

        <output message="InformationMessage"/>

    </operation>

</portType>
```

Figure 8.12: WSDL portType specs for third party services

```
<process name="SESPProcess">

    <partnerLinks>

        <partnerLink name="transmission"

          partnerLinkType=""

          myRole="ReceiveMsgSrv" />
          .
          .
          .
    </partnerLinks>

    <faultHandlers>
     .
     .
     .
    </faultHandlers>
```

Figure 8.13: Example SESP Process (part I)

```
<sequence>
    <flow>
        <sequence> // New message from sender
            <receive partnerLink="transmission"
              portType="SESPReceiveMsgPT"
              operation ="SendMessage" variable ="M">
            </receive>
        </sequence>
        <sequence> // Refined message retransmission
            <receive partnerLink="self-transmit"
              portType="SESPReceiveMsgPT" operation ="SendMessage" variable ="M">
            </receive>
        </sequence>
        <sequence> // Call back service
            <receive partnerLink="RESP-SESP-CB">
              portType=" SESPCallbackPT" operation="MessageCallback" variable="RefinementMsg">
            </receive>
            <invoke partnerLink="self">
              portType=" SESPImprovementPT">
              operation="MessageCallback" inputVariable="RefinementMsg" outputVariable="M">
              <throw "FailureFault" faultVariable="RefinementMsg">
            </invoke>
            <reply partnerLink="self-transmit">
              portType="SESPReceiveMsgPT" operation="SendMessage" variable="M">
            </reply>
        </sequence>
```

Figure 8.14: Example SESP Process (contd.) (part II)

```
<flow> // message preparation

  <links>

    <link name="fix-deliver"/>

    <link name="UDDI-resn">

  </links>

  <sequence>

    <invoke partnerLink="scanner"

      portType=" SESPExaminationPT"

      operation=" VirusScan" inputVariable="M" outputVariable="result">

    </invoke>

    <switch>

      <case condition="result=true">

        <invoke partnerLink="scanner"

          portType="SESPVirusRemovalPT"

          operation="VirusRemoval" inputVariable="M" outputVariable="M">

          <source linkName="fix-deliver" />

        </invoke>

      </case>

      <otherwise>

        <empty />

      </otherwise>

    </switch>

  </sequence>
```

Figure 8.15: Example SESP Process (contd.) (part III)

```
          <sequence> // where to send?
            <invoke partnerLink="nameReslv" portType="UDDIService"
              operation="GetAddress" inputVariable="From" outputVariable="IPAddress">
              <source linkName="UDDI-resn" />
            </invoke>
          </sequence>
          <sequence> // send message to RESP
            <invoke partnerLink="outbound"
              portType="SESPConnectPT" operation name="GetSLA"
              inputVariable="M" outputVariable="SLA">
              <target linkName="UDDI-resn" /> <target linkName="fix-deliver" />
            </invoke>
            <while condition="number &lt; SLA">
              <flow>
                <sequence>
                    <invoke partnerLink="destination" portType="SESPDeliveryPT"
                      operation name="SendMessage" inputVariable="M" outputVariable="R">
                      <catch "RejectionFault" faultVariable="RejectNotice">
                        <reply partnerLink="SenderCB">
                          portType=" SenderMsgCallbackPT" operation="Rejection" variable="RejectNotice">
                      </catch>
                    </invoke>
                </sequence>
            </while>
          </sequence>
        </flow>
    </sequence>
</process>
```

Figure 8.16: Example SESP Process (contd.) (part IV)

Table 8.4: Acceptance Policy

SLA and AP policies			
SLA	Allow		10 messages per connection
	Allow		Feedback for rejected messages
AP 1	Accept	IF	No virus/worm is attached
		AND	Filter allows receipt
			OR
			Distributed checksum allows receipt
AP 2	Accept	IF	No virus/worm is attached
		AND	Message bonded with value $\geq$ b
		AND	Bond is verified by an escrow service

As shown in figure 8.17, upon invocation of RESP's RecieveMsgPT ("Get-Message" operation) the message is transmitted to RESP. For each received message, the RESP applies a message acceptance policy to accept or reject it. If the transmitted mail fails to satisfy this policy, the RESP either throws a rejection notice or a refinement message. The refinement message suggests changing some parts of the message that may make it acceptable to the RESP. As a result, refinement activity may begin at the SESP. Note that based on its own policy, an SESP may decide to ignore all advice, and consequently, the callback service interface may not be exposed (the current strategy used by existing SMTP implementations). On the other extreme, if neither party stops the refinement process, it may go on forever. Many such strategies have been studied by researchers in other contexts (like automated trust negotiation [68], *etc.*), and can be supported here. In the code presented next, we take the approach of refining a message up to a specified number of times (5 here). This is because we haven't found the need yet for a more complex strategy.

The RESP process is made up of five main parts, as shown in figures 8.17, 8.18, 8.19, 8.20 and 8.21, viz, headers, types and supported faults (figure 8.17), message

```
<process name="RESPProcess">

    <partnerLinks>

        <partnerLink name="ESPtransmission" partnerLinkType="" myRole="ReceiveMsgSrv" />

        .

        .

        .

    </partnerLinks>


    <faultHandlers>

        .

        .

        .

    </faultHandlers>


    <sequence>

   // logic for generating SLA

      <switch> // Evaluate SLA

        <case condition="number &lt; 11">

          <receive partnerLink="RESPtransmission"

            portType="RESPReceiveMsgPT" operation ="GetMessage" variable ="M">

          </receive>
```

Figure 8.17: Example RESP Process (part I)

```
<flow> // Invoke concurrent processes
  <links>
  </links>
  <sequence> // Virus scanning
    <invoke partnerLink="scanner" portType="RESPExaminationPT"
      operation=" VirusScan" inputVariable="M" outputVariable="result">
    </invoke>
    <switch>
      <case condition="result=true">
          <invoke partnerLink="scanner" portType="RESPVirusRemovalPT"
            operation="VirusRemoval" inputVariable="M" outputVariable="M">
            <source linkName="fixed" />
          </invoke>
      </case>
      <otherwise>
        <empty>
          <source linkName="empty" />
        </empty>
      </otherwise>
    </switch>
  </sequence>
  <sequence> // Distributed checksum
      <invoke partnerLink="TPDCC"" portType="CheckSumPT"
        operation=" DCC" inputVariable="M" outputVariable="checksumOK">
        <source linkName="dcc-deliver" />
      </invoke>
  </sequence>
```

Figure 8.18: Example RESP Process (contd.) (part II)

```
        <sequence> // Verify bond
            <invoke partnerLink="TPEscrow"" portType="bondVerificationPT"
              operation="VerifyBond" inputVariable="M" outputVariable="verified">
              <source linkName="bond-verify" />
            </invoke>
        </sequence>
        <sequence> // Bayesian filtering
            <invoke partnerLink="RESPFilter"" portType="RESPFilterPT"
              operation="BayesianFiltering" inputVariable="M" outputVariable="filterOK">
              <source linkName="filtering" />
            </invoke>
        </sequence>
      </flow>
<!- enforcing acceptance policy -->
      <sequence>
        <switch>
            <case condition="(fixed OR empty) AND (checksumOK OR filterOK))">
              <invoke partnerLink="RESP-Recipient" portType="RESPStoragePT"
                operation="StoreMessage" inputVariable="M"
                <switch>
                  <case condition="Ack = YES" outputVariable="delivered">
                  </case>
                  <case condition="No space">
                    <throw "FailFault"> </case>
                  <otherwise> <empty /> </otherwise>
                </switch>
              </invoke>
            </case>
```

Figure 8.19: Example RESP Process (contd.) (part III)

```
<case condition="(fixed OR empty) AND (verified AND bond &gt; b)">
  <invoke partnerLink="RESP-Recipient" portType="RESPStoragePT"
    operation="StoreMessage" inputVariable="M"
    <switch>
      <case condition="Ack = YES">
        outputVariable="delivered">
      </case>
      <case condition="No space">
        <throw "FailFault">
      </case>
      <otherwise> <empty />
      </otherwise>
  </invoke>
</case>
<case condition="NOT fixed OR NOT (checksumOK AND filterOK)>
  <throw "RejectionFault" faultVariable= "RejectionNotice">
  </throw>
</case>
<otherwise>
  <sequence>
    <switch>
      <case condition="history &gt; 5">
      // 5: maximum invocations of improvement service
        <invoke partnerLink="self" portType="RESPImprovementPT">
          operation="RefineMsg" inputVariable="M" outputVariable="RefinementMsg">
          // outputVariable stores M's refinement history
        </invoke>
```

Figure 8.20: Example RESP Process (contd.) (part IV)

```
                        <reply partnerLink="RESP-SESP-CB" portType=" SESPCallbackPT">
                          operation="MessageCallback" Variable="RefinementMsg">
                        </reply>
                      </case>
                      <otherwise> <empty />
                      </otherwise>
                </otherwise>
            </switch>
          </sequence>
     </switch>
    </sequence>
 </process>
```

Figure 8.21: Example RESP Process (contd.) (part V)

reception from SESP (figure 8.18); invocation of helper services to estimate message quality (figures 8.18 and 8.19); acceptance policy evaluation based on message quality (figures 8.19 and 8.20) and finally, computing feedback for rejected messages (figure 8.21). The RESP waits for messages to arrive, and if the service level agreement is satisfied, messages are accepted (as shown in the process listings). Next, the RESP makes concurrent calls to several 'helper' services, like Bayesian filtering service, bond verification service, distributed checksums, virus scans, etc., to estimate the quality of incoming message. Once these processes terminate with an output, the RESP process starts evaluating the message based on its acceptance policy. During this stage a message may be accepted or rejected. Rejected messages may be returned to the SESP with feedback on some hints usable for resubmission, if the sender decides to do so (using the message improvement service).

Example 14 (Message rejection by RESP)**.** *Assume a mail message (M) that contains the following appropriately initialized parts: From, To, Date, ID, Subject and Body. We make the following assumptions:*

- *M does not contain any attached virus/worm*
- *M is the only message in queue*
- *RESP's SLA accepts 10 messages per connection, and provides feedback for rejected messages.*
- *Acceptance policy requires that no virus be attached to a message, and either the message has a bond ("Surety") or satisfies the Bayesian filter.*
- *Message content may contain prohibited words.*

According to the generic BPEL processes described, with the change that above policy instead of the one shown in table 4 is evaluated, M will not be accepted for delivery at the RESP (figures 8.19 and 8.20). This is because it fails to satisfy both conditions - it doesn't include a valid bond and it doesn't satisfy the Bayesian filter on account of the prohibited words in its body. As in figure 8.20, the RESP process initiates a call to the message improvement service (to allow the sender to revise the message). The content of the refinement message would include the following parts: Date, ID, Sign, Surety and Body - the missing information that caused rejection. Essentially, this response provides the sender acceptable values for the parts Date, ID, Surety and Body. That is, the refinement message identifies the deficiencies in M: no valid bond (or surety) and presence of prohibited words. Once made aware, the sender may choose to alter the rejected message, so that it reaches its destination [66].

Chapter 9: Related Work

9.1 Introduction

The main areas of research in the email environment are primarily restricted to solving the delivery decision problem. This is in response to the omnipresence of *spam* or unsolicited commercial email witnessed by the majority of email users, as reported by Dwork and Naor [27] and by Bass, Freyre *et al.* in [9]. The latter work mainly points out the inadequacies of the current infrastructure through which malicious senders or spammers can abuse it by sending spam. However, an important aspect of the problem they present has been largely overlooked, namely, the delivery scheduling problem wherein a sender can overwhelm the recipient mail system by sending a large amount of mail messages. These articles [27, 9] lay the foundation of our study. Other related work which have influenced the formulation and direction of this work are classified into five broad genres: *a)* Laws and legislative procedures; *b)* Content based filtering and text categorization solutions; *c)* Reputation and trust based solutions; *d)* Economic solutions enforcing cost or memory resources and *e)* Network based solutions. Next we present most significant related work in two broad classes: The solutions which have been implemented and are a part of the current industry practice and proposed solutions which have not yet been implemented. Within each category we subclassify the approaches according to their genre and relate them to our approach.

9.1.1 Proposals

Legal Provisions

At one front of solutions against fraudulent email or spam are *laws and legislative provisions*. Laws are essential and helpful but certainly not the silver bullet we have been searching for, as summed up in [69]. The main limiting aspect of legal solutions is that they are limited within their jurisdictions but spammer's have global reach. Hence spammers out of a country can break laws against spam without fear of legal action. Several countries have drafted such laws (For example Can Spam act in the United States *etc.*), however, the definition of spam and evidence of spamming are issues of contention. Again, in the absence of international agreements, such approaches will not have any major impact. Laws without technical support can be worthless and ill-directed. Some techniques have been suggested to gather credible evidence of spamming. Recently a project called 'Project Honeypot' [70] was initiated, which claims to provide credible third party evidence against individuals breaking the laws against spam. Email filters that screen out illegitimate mails can also be used to gather evidence against spammers [71]. Investigators have pointed out various techniques used by spammers as a means to further their illegitimate activities [72]. Such initiatives are a step in the right direction and can be combined with reputation or trust based solutions to protect the recipients against illegitimate mail. In our approach, we provide means to consolidate such claims by providing irrefutable evidence of the communication.

Content based filtering and text categorization solutions

Content based filtering solutions try to parse the content of a message and determine the probability of it being spam. Cohen's work [73] was among the first to suggest text categorization for automated filtering applicable in many applications, followed by [74] for specific use in email applications. Since then a lot of approaches have been put forth for filtering purposes, some of earlier ones are [75] using support vector machines algorithm, LogitBoost algorithm[76] *etc.* Sahami, Dumais *et al.* [77] first applied bayesian classification techniques to email filtering in a decision theoretic framework and showed the efficacy of such filters. Naive Bayesian filters [78, 37] provide a probability of the text classification decision and quantify the probabilities of spam determination. Yerazunis presented a high performance technique called sparse binary polynomial hashing [30], a generalization of the Bayesian method that can match mutating phrases as well as single words. Chhabra, Yerazunis *et al.* [79] also propose to use Markovian discrimination with variable neighborhood windows for features in spam filtering. This is essentially the marriage of sparse binary polynomial hashing with feature weighing and it is shown that much better filtering results have been achieved. Other statistical approaches include Filtron [6] which uses machine learning to construct effective personal anti-spam filters by better feature engineering; study on support vector machines and random forests [80] show that they can result in lower false positive rates than the naive bayesian spam filtering. Chhabra, Yerazunis *et al.* [81] present a non-probabilistic statistical approach to spam filtering and have achieved a higher success in reducing the error rates. In this context, the work by Wittel and Wu [82] studied various attacks on statistical spam filters and shown techniques that could be used against these filters. From our perspective, advances in statistical filtering are extremely important and we leverage on their techniques to design a comprehensive mechanism to approach the delivery decision problem. Our research is orthogonal, or at best complementary to these proposals as we provide a better infrastructural support for filtering in our approach.

We also cover here some of the non-statistical filtering methodologies proposed. Rigoutsos and Huynh [83] propose a pattern matching algorithm based on the Teiresias pattern discovery algorithm for spam filtering. However, their algorithm has not been tested against statistical spam filters. Kolcz *et al.* use a signature driven algorithm in [84] to detect near-duplicate spam messages. The authors assume that most spam messages are near-duplicates and hence the applicability of their proposal. Leiba and Borenstein note in [85] that elimination of spam is impossible and the best way to approach towards the spam problem is to use multiple solution types to filter spam messages. In fact, their approach is very close to our proposal to repair the ailing email system and can be seen as one particular delivery mechanism $\mathcal{M}$ determined by the set $\mathcal{C}$, the sequence of spam reduction approaches suggested in this paper. Gray and Haahr note in [31] that collaborative filtering is not ideal for individual users and present a new architecture called CASSANDRA for personalized collaborative spam filtering. This system tries to push back the filtering process closer to the source, though still maintaining the properties that only legitimate messages (relevant to the user) are stopped. One of the distinguishing features of this work is that unlike other filtering approaches, where messages are tagged as spam without individual recipient's opinion, this work does not make such assumptions. With respect to our work, this approach is highly relevant and we leverage on their architecture and generalize it through our policy aggregation technique to be presented later. Yerazunis *et al.* suggest a unified model of filters in [86] independent of the learning algorithm used. Such a technique would help benchmark spam filtration technology in a standard way. A word stemming approach by Ahmed and Mithun is proposed in [87] which attempts to discover stems of word mutations to increase the efficacy of statistical filters. Similar to this approach, Oliver suggests in [88] use of lexicographical distances to discover morphed expressions. In [89] Zdziarski suggests the use of a 'pre-filter' called bayesian noise reduction to improve the data processing of a statistical filtering approach.

Content based spam filtering is one of the most popular approach to solve the delivery decision problem and has proved to be an effective approach against receiving unwanted email messages. However, one of its main drawbacks is the fact that a mechanism determined by spam filters does not make an effective delivery mechanism. Messages are stopped close to the destination and as a result, the bandwidth utilization of the email system is reduced. In fact, since the use of spam filters, number of illegitimate mail messages have drastically increased to more than half of the email bandwidth. The cost of handling these messages and transmitting them close to the destination is borne by the infrastructure. It is clear that filters alone cannot determine an effective delivery mechanism, and some additional properties must be used so that unwanted messages are stopped as close to the source as possible.

Economic solutions

Economic solutions address the question of costs incurred in solving the delivery decision problem. Allman in [90, 69] noted that spam filtration technology puts the complete burden of costs of the solution on the recipient and the recipient mail system. He advocates designing solutions such that costs can be shared between the recipients and the senders and thus mount an effective deterrent to unwanted mail. In fact a mechanism based on this principal can potentially be an effective mechanism if it stops messages close to the source. *Information Asymmetry* approach by Loder *et al.* [3] describes a new paradigm for spam mitigation based on monetary signalling, which aims to increase email costs primarily for senders who misuse the email infrastructure. The paper builds on the concepts of interrupt rights [91] to do so and can charge message senders money for recipient attention span, if the recipient chooses so. For unwanted messages, a recipient can thus get compensated for his or her time spent in dealing with the message. In their view, messages bonded above a threshold reveal the confidence of the sender in the message content and can help a recipient safely determine the utility of the message for him or her. With respect to monetary transactions, they claim their scheme to be policy independent and is based on configuring a minimum monetary threshold for attention span. This proposal is an excellent technique for a proactive charge against spam as it aims to reduce the bandwidth unwanted messages use up. In our work, we can accommodate message bonds and allow them to be used with other delivery decision solutions. We achieve this goal through a policy framework and hence our work is more general. Rui and Li in [32] argue that differentiated surcharge mechanism using a combination of cost and filtering approaches is better than bonds or filters alone. Instead of letting a recipient choose a monetary threshold as proposed by Loder *et al.*, the authors suggest using filters on every mail message and the probability of it being spam determines the price charged to the sender. Since they propose use of personalized spam filters, indirectly the recipient chooses varying levels of monetary costs levied on the senders, based on the 'spamminess' of messages according to him or her. Empowering the user to control quality of emails is great, but it may lead to abuse of the system by recipients.

Another paradigm for mail delivery is defined by Templeton in [29] and Kraut *et al.* in [24]. They advocate a static fee for delivering mail, something akin to postage in regular mail services. Hence, for every message a sender pays a nominal fee, and in case of spammers, such a fee would amount to a large sum of money and hence will be discouraged from generating large volumes of spam. These solutions can drastically reduce spam bandwidth and could in turn make the spam filters extremely accurate (due to low bandwidth of spam messages), but every one will have to bear the cost of sending email and it is against the open internet flavor. In general, we would like the email service not to punish legitimate users for misbehavior of others. However, it is still an excellent means of addressing the delivery decision problem and we allow for its use in our framework. In our framework, if a sender really wants to get through and is ready to pay the price set by the receiver, will get through. A computational counterpart of monetary schemes involves use of 'puzzles' or computation intensive calculations [27] or memory based costs [26] to shift the computational burden of costs resulting from spam to the senders. This solution is attractive because there is a possibility of reducing unwanted messages without senders incurring monetary costs. Balakrishnan and Karger in [25] suggest use of quota allocation and enforcement to limit number of messages sent by a sender.

Reputation and trust based solutions

Trust based solutions grant or deny delivery access to message requests based on delivery rights. In the networking parlance, trust based systems, are said to use whitelists and blacklists. A whitelist is a set of senders who are given the right to deliver to the whitelist author's mailbox, whereas blacklist is a set of senders from whom the delivery right has been revoked. For senders not in either list, the delivery right has not yet been granted or revoked, but must be earned by satisfying terms or conditions set by the recipient. There are several issues involved in these types of systems. First and foremost is establishing the identity of a sender without which the access control system is unenforceable. Email sender authentication is not a trivial task as identity spoofing is relatively easy. Several methods have been proposed to get around the spoofing problem; we review some of these approaches next. Source authentication using digital signatures for every email message has been suggested in [5]. Although strong sender authentication would be helpful but the scalability of this scheme is doubtful. Pretty Good Privacy (PGP) (email confidentiality/privacy solution) was shown to falter because of scalability reasons, as large scale PKI deployments are not scalable and web of trusts can be attacked by setting up malicious web rings [92]. Also, the ease of spawning of new digital identities [21] in an open environment such as internet is an important issue to be addressed. A weaker source authentication scheme proposed by Boykin and RoyChowdhury [93] takes advantage of the social networks to construct a list of trusted senders using depth of communication hierarchies. The authors only achieve limited success with this approach and note that this technique alone is insufficient to catch all spam in email. Potentially spammers and intruders can attack this system by monitoring clear text messages to gain unrestricted access to a mailbox. Here we also mention a network based solution called SPF [94] that aims to solve the spoofing problem. We cover it in more detail in the next section. In all, attempts are being made to provide sender authentication for email, such that trust based solutions can be used for the delivery decision problem. Solving the spoofing problem is clearly not enough as shown by Watson in [95]. He argues that given reliable sender authentication is available, policies based on whitelists and blacklists can still be circumvented due to very large number of email addresses as well as the ability to attain a new email address with much ease.

The second issue with trust based systems is the construction and revisions of whitelists or blacklists, also known as ACLs, *i.e.*, access control lists. The problem is stated as follows: recipients must be able to state a set of conditions, which once satisfied, allow a recipient to place their trust in the requesting party. Usually, recipients are expected to construct ACL's manually but new approaches have been proposed to automate this process. *Challenge Response* solutions [4] which employ passwords or captcha tests are one instance of such a proposal. The basis of this test is that a human is involved in the loop, and hence can be trusted for sending a legitimate message. Since a majority of spammers use automated programs to generate large volumes of messages when spewing out spam, the presence of a human in the loop seems to be a reasonable reason to accept the message. Once senders authenticate themselves, their email addresses are added to the recipient's whitelist. However, this solution is not error proof and potential attacks can be constructed by eavesdropping and harvesting trusted senders for targeted recipients. Once a sender is in a whitelist, he or she gets full access to the recipient's mailbox, and hence the vulnerability. Similarly, on failing to pass the test posed by a recipient may lead to the sender being accorded blacklisted status. It may be possible to cause denial of email service to a particular sender for sending messages to a particular recipient by engineering an attack that blacklists the sender in the recipient's ACL. Therefore, use of challenge response in isolation is unsatisfactory and needs to be combined with an authentication mechanism for more stable results. The problem of maintenance or revision of ACLs has not been explored by this technique.

Reputation based systems build trust through associations or referrals. Goldbeck *et al.* in [35] suggest the use of a reputation based system which builds trust through association. Their prototype system called TrustMail is very similar in approach to [93] but instead of populating a whitelist or using the connection hierarchy, they use user reputation ratings to determine the utility of mail messages. In a sense they achieve ratings corresponding to whitelists (though dynamic in nature) and can sort messages based on the aggregate reputation of the sender. This technique is closely related to our approach where we aim to build reputation systems based on associations or referrals depending upon requirements at the recipient end. Project Lumos [34] is another reputation based system, which includes the notion of two-level sender authentication supported by public reputation system, similar to what our proposal assumes. Lumos is unique in the sense that they use predefined *signals* as message headers to help the recipient solve the delivery decision problem. Message characterizations by senders may include 'Urgent', 'Official', 'Advertisement' *etc.* and the recipient can write filters to sort the received mail. Assertions by senders are supported by a central public reputation network which collects complaints against sender's message characterizations and can be used to derive the trust level at the recipient. The limitation of Lumos is that it is set up, at heart, to serve the interests of the large commercial bulk mailers. The idea is to register bulk emailers, require them to respond to reports of abuse, and publish the corresponding reputations. If Lumos really does make headway in the marketplace, our scheme would be easier to implement, since we rely on the same notions of sender authentication. Also, Lumos would be an excellent reputation service to include in our framework which is based on distributed reputations. Other examples of proposed or existing reputation services are Razor [40] and CloudMark [22], which aim to identify individual SPAM messages through a collective effort, and SmartScreen [96], which is a Microsoft effort with the same objective. Both Razor and SmartScreen would fit well with the approach that we propose.

Network based solutions

We have categorized methods as network based if they depend heavily on network studies. Some of the methods discussed under this genre may as well belong to one or more genres discussed above and some works already discussed above reappear in this section. We begin our survey with the work of Li *et al.* who propose to introduce delays in the transport layer during mail transmission in [28] to slow down a spammer. Authors contend that an early detection mechanism using spam filters can be used to determine connections that should be delayed. By doing so, the delivery rate of malicious senders can be significantly reduced, whereas the legitimate senders would experience negligible delays. Thus by increasing the time to transfer a message, TCP-damping increases the cost of sending messages to spammers. However, the recipient mail system also pays an equivalent cost by allocating costly TCP sessions to unwanted mail sessions. Though novel, we feel that this method needs more finesse. In our work, we use a similar approach to identify unwanted or low priority mail delivery sessions and allow use of policies to restrict the number of messages transferred in each such session, hence achieving almost similar results as presented in this work.

Sender policy framework proposed by Wong [94], and earlier approaches like reverse DNS checks [97, 98] propose to solve the spoofing problem. They try to plug the current vulnerability in the email infrastructure by which a sending mail system is allowed to assume any mail domain name. In this approach, the originating IP-address and the presented domain name are matched through reverse DNS queries of MX (mail exchange) records. When the two match, the recipient mail system is convinced of the identity of the sender, otherwise, the recipient can choose not to trust the sender with their mail messages. A mechanism for sender authentication is badly required for establishing trust between internet hosts, senders and recipients and we leverage on this methodology heavily in our work.

Realtime Blackhole Lists (RBLs) [36] maintain a list of IP addresses which have been used by spammers or sending mail systems to spew out spam. These lists serve as a reputation service employed by recipient mail systems to evaluate amount of trust in the communicating party. Spam URI Realtime blocklists (SURBL) [46] are used to detect spam based on message body URIs. Unlike most other RBLs, SURBLs are not used to block spam senders. Instead they allow recipients to block messages that have spam hosts which are mentioned in message bodies. In our work, we allow the use of reputation services of this nature to leverage on their work. Razor [40] or Cloudmark [22] and Distributed Checksum Clearinghouse [23] similarly help stop near-duplicate spam messages by maintaining a count of duplicates seen on the internet and flagging those which cross a threshold. In our approach we allow use of such IP and message reputation services to be used while making the delivery authorization.

Clayton proposes to use extrusion detection for stopping spam in [33]. The proposal is to use extrusion filters, similar to incoming mail filters at the service provider level. If a large number of users use the same email gateway (more details in chapter 3), then potential spammers and captured machines can be stopped from spewing out spam and thus protecting target victims of spam attacks. The mechanism resulting from use of extrusion detection can stop messages closer to the source and prove to be more effective than many other methods. Moreover, using extrusion detection will also protect innocent boxes and domains from getting blocked by RBLs. In our approach, we keep this requirement in mind and propose an architecture where such filters can be employed.

Dai and Li in [38] present a new approach using third party mail arbiter to test the sent message against a particular recipient's preferences. If the message does not meet the recipient's criteria, the online service contacts senders and gives them a reason for rejection of the message, in turn allowing the sender to *fix* or customize the message for recipient's preferences. This work is very similar to our approach where we include the feedback mechanism in the current infrastructure itself, making the use of third party redundant.

9.1.2 Current practices

Haskins and Nielson in [99] provide an excellent introduction to some of the products used for filtering emails. Products covered are: ProcMail [100] and SpamAssassin [7] filtering extensions in Sendmail, Postfix and qmail; SMTPAUTH [19] and STARTTLS [20] configurations in Sendmail, Postfix and qmail; Distributed checksum filtering solutions like Vipul's Razor [40]and DCC [23]; bayesian filtering methods like CRM114 [30], bogofilter [101] and ASSP [102]; email client filtering products POPFile, Mozilla Messenger, Microsoft Outlook. Procmail can be configured to perform actions based on patterns matched in header items as well as the body of a mail message. Actions include filing, forwarding, and further processing. Typically it is used when the desired anti-spam program requires it or when no other mechanisms are available. For example, bogofilter and blacklists can be used within Procmail. Spamassassin is a widely used interface to other anti-spam mechanisms as well as a spam-classifier itself. Methods used in Spamassassin include message analysis(header and body), bayesian filtering, distributed checksums, RBLs, automatic and manual whitelisting/blacklisting. A natural descendent of static whitelists and blacklists are blackhole lists (RBLs). Some of avaiilable RBL services are MAPS [103], SpamCop [104] and Spamhaus [105]. RBLs can be easily integrated with the mail server software. STARTTLS amd SMTPAUTH are extensions to the current mail system to support strong authentication and are in use now. Market currently abounds in bayesian filtering products, with more than 80 products mentioned in [99]. Academic proposals like [30] are now included as bundled software packages in many anti-spam solutions. CRM114 is reputed to be highly accurate as is the bogofilter. ASSP has a very easy to configure GUI and hence a good choice. Email client filtering is more accurate than collaborative filtering since it customizes filters for individual use. Many such products are available for use.

Similar to a monetary and computational or memory based cost solutions, Internet

Mail 2000 [106] software proposes to use sender memory resources for all unread mail and hence reduce the amount of spam. Recipients receive mail notifications and can choose to read or delete the message. However, it does not effectively solve the delivery decision problem, but does stop messages at the source, thus improving the bandwidth utilization.

Chapter 10: Conclusion

Needs at one recipient mail system could be starkly different from another depending upon the type of organization. For example, a university mail system may not be as restrictive as a corporate mail system which in turn may have different requirement than a defence organization. The abundance of solutions to protect against unwanted mail messages provides ample choices for recipients to pick and choose from, to get a combination which satisfies their particular needs. As noted above, anti-spam solutions used in practice usually employ more than one anti-spam technique. However, due to combination of methods, information leakage channels may arise, which can lead to compromise of sensitive information. In this thesis we have provided a means for picking and choosing the desired set of email-control techniques and using them under a single umbrella. In addition, we provide a means to protect against inadvertent information leakages as well.

Other related work in the email domain, not directly related to delivery decision problem, have also influenced this thesis. For example, *Certified Email* [107, 108, 109, 110, 111, 112, 113] mainly addresses the problem of confidentiality and fair email exchanges involving a trusted third party. The problem space is not completely disjoint with the spam problem, since monetary solutions will definitely need fair exchange and protect senders against abuse by recipients. We hope to take advantage of some results in these works in future work. *Anonymous Email* [114, 115, 116, 117, 118] is potentially a casualty of the quest to solve the delivery decision problem.

Sender authentication is one of the preconditions to trust based solutions, but it could increase the difficulty of legitimate uses of anonymous messages, such as whistle blowing. Achieving strong anonymity is an extremely hard problem due to collusion attacks and network monitoring capabilities. In any case, strong anonymity requires a mechanism on top of the current mail delivery protocol and hence is orthogonal to the current research proposal. Nonetheless, an interesting research direction is *source authenticated anonymous mail. Distributed Reputation Management Systems* [119, 120, 121, 122, 41] are an indispensable tool to reward faithful principals and punish malicious ones. An interesting future work would involve looking into both the local and global beliefs about trustworthiness of a principal and how local beliefs can be distributed using deception resilient algorithms (applicable to the email domain).

We believe we have sufficiently addressed the claims made in this thesis (see chapter 1). Next we recap the claims in chapter 1 and the reason we come to the above conclusion of addressing these claims. In claim 1 we say that a majority of the existing email-control techniques can be supported in our policy-based framework. We show a Web-based implementation in chapter 8 that incorporates several email control techniques in a policy based framework to control email delivery. This framework can be easily extended to include new email-control techniques in future (possibly). Claims 2 and 6 state that feedback for email can be incorporated into email systems that will lead to reduced chances of losing desirable messages to email filtering. Chapters 4,5,7 and 8 present the design and instrumentation of a feedback system into conventional SMTP systems. In chapter 4 we show that with feedback system, message refinement, policy-based email delivery and advertisement of downstream policy requirements upstream, only messages that satisfy the downstream policy will get accepted downstream – this shows claim 3 and claim 4. Claim 5 says that downstream privacy leakages (upstream) can be prevented by *sanitizing* acceptance policies before communicating them upstream. This claim is shown in chapter 6.

Bibliography

Bibliography

[1] Simple Mail Transfer Protocol, "RFC 2821," Apr 2001.

[2] J. F. Kurose and K. W. Ross, *Computer Networking: A top down approach featuring the internet*, 2nd ed. Addison Wesley, 2003.

[3] T. Loder, M. V. Alstyne, and R. Wash, "Information asymmetry and thwarting spam," University of Michigan, Tech. Rep., 2004.

[4] M. Naor, "Verification of a human in the loop or identification via the turing test," http://www.wisdom.weizmann.ac.il/ naor/PAPERS/human_abs.html, 1996. [Online]. Available: citeseer.ist.psu.edu/naor96verification.html

[5] T. Tomkins and D. Handley, "Giving email back to the users: using digital signatures to solve the spam problem," in *First Monday* 8*,* 9 *in http://firstmonday.org/issues/issue*8_9*/tomkins/index.html*, September 2003.

[6] E. Michelakis, I. Androutsopoulos, G. Paliouras, G. Sakkis, and P. Stamatopoulos, "Filtron: A learning-based anti-spam filter," in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[7] Spamassassin, "http://useast.spamassassine.org/."

[8] A. K. Bandara, E. C. Lupu, J. Moffet, and A. Russo, "A goal-based approach to policy refinement," in *IEEE 5th International Workshop on Policies for Distributed Systems and Networks*, New York, June 2004.

[9] T. Bass, A. Freyre, D. Gruber, and G. Watt, "E-mail bombs and countermeasures: Cyber attacks on availability and brand integrity," *IEEE Network*, vol. 12, no. 2, pp. 10–17, Mar-Apr 1998.

[10] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian, "Flexible support for multiple access control policies," *ACM Transactions on Database Systems*, vol. 26, June 2001.

[11] F. Fages, S. Soliman, and R. Coolen, "CLPGUI: a generic graphical user interface for constraint logic programming," *Journal of Constraints*, vol. 9/4, 2004.

[12] Internet Message Format, "RFC 2822," Apr 2001.

[13] M. Pop, "Comparative study of electronic mail systems." [Online]. Available: citeseer.nj.nec.com/pop94comparative.html

[14] MIME (Multipurpose Internet Mail Extensions), "RFC 1521," Sept 1993.

[15] A. S. Tanenbaum and M. v. Steen, *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[16] Post Office Protocol (POP) Version 3, "RFC 1939," May 1996.

[17] Internet Message Access Protocol - Version 4, "RFC 3501," March 2003.

[18] Mail Routing and Domain Name System, "RFC 974," Jan 1986.

[19] SMTP Service Extension for Authentication, "RFC 2554," March 1999.

[20] SMTP Service Extension for Secure SMTP over Transport Layer Security, "RFC 3207," Feb 2002.

[21] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara, "Reputation systems," *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, December 2000. [Online]. Available: http://www.si.umich.edu/ presnick/papers/cacm00/reputations.pdf

[22] CloudMark's SpamNet, "http://www.cloudmark.com."

[23] D. C. clearinghouse, "http://rhyolite.com/anti-spam/dcc/."

[24] R. E. Kraut, S. Sunder, J. Morris, R. Telang, D. Filer, and M. Cronin, "Markets for attention: Will postage for email help?" in *CSCW'03: Computer Supported Cooperative Work*. New York: ACM Press, 2003, pp. 2006–215.

[25] H. Balakrishnan and D. Karger, "Spam-I-am: A Proposal for Spam Control using Distributed Quota Management," in *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[26] C. Dwork, A. Goldberg, and M. Naor, "On memory-bound functions for fighting spam," in *CRYPTO'03: Advances in cryptology*, 2003.

[27] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *CRYPTO'92: Advances in cryptology*, 1992.

[28] K. Li, C. Pu, and M. Ahamad, "Resisting spam delivery by tcp damping," in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[29] E-Stamps, "http://www.templetons.com/brad/spam/estamps.html."

[30] W. S. Yerazunis, "Sparse binary polynomial hashing and the CRM114 discriminator," in *2003 Cambridge Spam Conference Proceedings*, 2003.

[31] A. Gray and M. Haahr, “Personalized collaborative spam filtering,” in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[32] R. Dai and K. Li, “Shall we stop all unsolicited email messages?” in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[33] R. Clayton, “Stopping spam by extrusion detection,” in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[34] Email Service Provider Coalition, “Project Lumos,” http://www.networkadvertising.org/espc/lumos_white_paper.asp, Sept 2003.

[35] J. Golbeck and J. Hendler, “Reputation network analysis for email filtering,” in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[36] Realtime Blackhole List, “http://www.kelkea.com/.”

[37] P. Graham, “A plan for spam,” in *2003 Cambridge Spam Conference Proceedings*, 2003.

[38] R. Dai and K. Li, “Regulation instead of stopping,” in *Spam conference 2005*, Cambridge, Massachusetts, January 2005.

[39] T. Yu and M. Winslett, “Unified scheme for resource protection in automated trust negotiation,” in *Proceedings of IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, May 2003, pp. 110–122.

[40] V. V. Prakash, “http://razor.sourceforge.net/,” March 2000.

[41] B. Yu and M. P. Singh, “Detecting deception in reputation management,” in *Proceedings of Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003, pp. 73–80. [Online]. Available: http://www-2.cs.cmu.edu/ byu/papers/p112-yu.pdf

[42] S. Kaushik, P. Ammann, D. Wijesekera, W. Winsborough, and R. Ritchey, “A policy driven approach to email services,” in *IEEE 5th International Workshop on Policies for Distributed Systems and Networks*, New York, June 2004.

[43] T. Group, “The XSB Programming System,” http://xsb.sourceforge.net/.

[44] J. Jaffar and M. J. Maher, “Constraint logic programming: A survey,” *Journal of Logic Programming*, vol. 19/20, pp. 503–581, 1994. [Online]. Available: citeseer.ist.psu.edu/jaffar94constraint.html

[45] F. Fages, “Constructive negation by pruning,” *Journal of Logic Programming*, vol. 32/2, 1997.

[46] Spam URI Realtime Blocklist, “http://surbl.org/.”

[47] N. Li and J. C. Mitchell, "Datalog with constraints: A foundation for trust management languages," in *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003).* Springer, Jan. 2003, pp. 58–73.

[48] M. J. Maher, "A transformation system for deductive database modules with perfect model semantics," *Theoretical Computer Science*, vol. 110, pp. 377–403, 1993.

[49] T. C. Przymusinski, "On the declarative semantics of stratified deductive databases and logic programs," *Foundations of Deductive Databases and Logic Programming*, pp. 193–216, 1987.

[50] K. R. Apt, H. A. Blair, and A. Walker, "Towards a theory of declarative knowledge," *Foundations of Deductive Databases and Logic Programming*, pp. 89–148, 1988.

[51] M. C. Fitting, "A kripke-kleene semantics for logic programs," *Journal of Logic Programming*, vol. 2/4, pp. 295–312, 1985.

[52] D. Chan, "Constructive negation based on the completed databases," in *International conference on logic programming (ICLP)*, 1988, pp. 111–125.

[53] L. Wang, D. Wijesekera, and S. Jajodia, "A logic based framework for attribute based access control," in *2nd ACM Workshop on Formal Methods in Security Engineering (FMSE 2004)*, October 2004, pp. 110–122.

[54] S. Petry, "Port 25: The gaping hole in the firewall," in *Proceedings of ACSAC'02 Annual Computer Security Applications Conference*, Dec 2002. [Online]. Available: http://www.acsac.org/2002/case/thu-c-1030-Petry.zip

[55] S. Kaushik, W. Winsborough, D. Wijesekera, and P. Ammann, "Email feedbak: A policy-based approach to overcoming false positives," in *3rd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code (FMSE 2005)*, Fairfax VA, November 2005.

[56] H. Tamaki and T. Sato, "Unfold/fold transformation of logic programs," in *Proceedings of the Second International Conference on Logic Programming*, S.-A. Tarnlund, Ed., Uppsala, 1984, pp. 127–138.

[57] C. Pfleeger and S. Pfleeger, *Security in Computing*, 3rd ed. Prentice Hall, 2003.

[58] Microsoft Corporation's Penny Black Project, "http://research.microsoft.com/research/sv/PennyBlack/."

[59] T. Sato, "Equivalence-preserving first-order unfold/fold transformation systems," *Theoretical Computer Science*, vol. 105, no. 1, pp. 57 – 84, October 1992.

[60] A. Roychoudhury, K. N. Kumar, C. Ramakrishnan, and I. Ramakrishnan, "Beyond tamaki-sato style unfold/fold transformations for normal logic programs," *International Journal of Foundations of Computer Science*, vol. 13, no. 3, pp. 387–403, 2002.

[61] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," *ACM Computing Surveys*, vol. 33, no. 3, pp. 374–425, 2001.

[62] R. Reiter, "The predicate elimination strategy in theorem proving," in *Proceedings of the second annual ACM symposium on Theory of computing*, Northampton, Massachusetts, 1970, pp. 180–183.

[63] K. D. Lux, M. J. May, N. L. Bhattad, and C. A. Gunter, "Wsemail: Secure internet messaging based on web services," in *2005 IEEE International Conference on Web Services (ICWS 2005)*, 2005, pp. 75–82.

[64] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business process execution language for web services," http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, 2003.

[65] R. N. Afandi, "AMPol: Adaptive messaging policy based system," Master's thesis in computer science, University of Illinois at Urbana-Champaigne, 2005.

[66] S. Kaushik, W. Winsborough, D. Wijesekera, and P. Ammann, "Email feedback: A policy-based approach to overcoming false positives," in *3rd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code (FMSE 2005)*, Fairfax, VA, 2005, pp. 73–82.

[67] ——, "Policy transformations for preventing leakage of sensitive information in email systems," in *20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Sophia Antipolis, France, 2006.

[68] T. Yu, X. Ma, and M. Winslett, "PRUNES: an efficient and complete strategy for automated trust negotiation over the internet," in *ACM Conference on Computer and Communications Security*, 2000, pp. 210–219.

[69] E. Allman, "Spam, spam, spam, spam, spam, the ftc and spam," $http : //www.acmqueue.com/modules.php?name = Content\&pa = showpage\&pid = 66$.

[70] Unspam, LLC., "http://www.projecthoneypot.org/," January 2005.

[71] Jon Praed, “You’ve Got Jail: Some First Hand Observations from the Jeremy Jaynes Spam Trial,” in *Spam conference 2005*, January 2005.

[72] Brian McWilliams, “Spam Kings,” in *Spam conference 2005*, January 2005.

[73] W. Cohen, “Learning to classify English text with ILP methods,” in *Advances in Inductive Logic Programming*, L. De Raedt, Ed. IOS Press, 1996, pp. 124–143. [Online]. Available: citeseer.nj.nec.com/cohen96learning.html

[74] ——, “Learning to classify Email,” in *1996 AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[75] H. Drucker, D. Wu, and V. N. Vapnik, “Support vector machines for spam categorization,” *IEEE transactions on Neural Networks*, vol. 10, no. 5, pp. 1048–1054, September 1999. [Online]. Available: http://www.site.uottawa.ca/ nat/Courses/NLP-Course/itnn_1999_09_1048.pdf

[76] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: a statistical view of boosting,” *Annals of statistics*, vol. 28, no. 2, pp. 337–374, 2000.

[77] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, “A bayesian approach to filtering junk E-mail,” in *Learning for Text Categorization: Papers from the 1998 Workshop.* Madison, Wisconsin: AAAI Technical Report WS-98-05, 1998. [Online]. Available: citeseer.nj.nec.com/sahami98bayesian.html

[78] P. Graham, “Better bayesian filtering,” in *2003 Cambridge Spam Conference Proceedings*, 2003.

[79] S. Chhabra, W. S. Yerazunis, and C. Siefkes, “Spam filtering using a markov random field model with variable weighting schemas,” in *ICDM’04: Fourth IEEE International Conference on Data Mining*, To appear 2004.

[80] G. Rios and H. Zha, “Exploring support vector machines and random forests for spam detection,” in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[81] C. Siefkes, F. Assis, S. Chhabra, and W. S. Yerazunis, “Combining winnow and orthogonal sparse bigrams for incremental spam filtering,” in *ECML/PKDD 2004: 15th European Conference on Machine Learning and 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2004.

[82] G. L. Wittel and S. F. Wu, “On attacking statistical spam filters,” in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[83] I. Rigoutsos and T. Huynh, “Chung-kwei: A pattern-discovery-based system for the automatic identification of unsolicited e-mail messages (spam),” in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[84] A. Kolcz, A. Chowdhury, and J. Alspector, "The impact of feature selection on signature-driven spam detection," in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[85] B. Leiba and N. Borenstein, "A multifaceted approach to spam reduction," in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[86] W. S. Yerazunis, S. Chhabra, C. Siefkes, F. Assis, and D. Gunopulos, "Unified model of spam filteration," in *Spam conference 2005*, Cambridge, Massachusetts, January 2005.

[87] S. Ahmed and F. Mithun, "Word stemming to enhance spam filtering," in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[88] J. Oliver, "Using lexigraphical distancing to block spam," in *Spam conference 2005*, Cambridge, Massachusetts, January 2005.

[89] J. A. Zdziarski, "Bayesian noise reduction: Progressive noise logic for statistical language analysis," in *Spam conference 2005*, Cambridge, Massachusetts, January 2005.

[90] E. Allman, "The economics of spam," $http : //www.acmqueue.com/modules.php?name = Content\&pa = showpage\&pid = 108$.

[91] S. Fahlman, "Selling interrupt rights: a way to control unwanted e-mail and telephone calls," *IBM Systems Journal*, vol. 41, no. 4, pp. 759–766, November 2002.

[92] P. Gutmann, "PKI: It's not dead, just resting," *IEEE Computer*, vol. 35, no. 8, pp. 41–49, Aug 2002.

[93] P. O. Boykin and V. Roychowdhury, "Personal Email Networks: An Effective Anti-Spam Tool," February 2004.

[94] Sender Policy Framework, "http://spf.pobox.com."

[95] B. Watson, "Beyond identity: Addressing problems that persist in an electronic mail system with reliable sender identification," in *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[96] Microsoft Corporation's SmartScreen Press Release, "http://www.microsoft.com/presspass/ press/2003/nov03/11-17ComdexAntiSpamPR.asp."

[97] Danisch.de: Defense against spam and E-Mail forgery, "http://www.danisch.de/work/security/antispam.html."

[98] M. R. T. C. F. R. Records, "*http : //www.mikerubel.org/computers/rmx_records/*."

[99] R. Haskins and D. Nielsen, *Slamming Spam: A guide for system administrators.* Addison Wesley, December 2004.

[100] Procmail, "ftp://ftp.procmail.net/."

[101] Bogofilter, "http://bogofilter.sourceforge.net."

[102] ASSP, "http://sourceforge.net/projects/assp."

[103] MAPS, "http://mail-abuse.org."

[104] Spamcop, "http://www.spamcop.net/vl.shtml."

[105] Spamhaus, "http://www.spamhaus.org/SBL/."

[106] Internet Mail 2000, "http://cr.yp.to/im2000.html."

[107] M. Abadi and N. Glew, "Certified email with a light on-line trusted third party: design and implementation," in *Proceedings of the eleventh international conference on World Wide Web (WWW'02).* Honolulu, Hawaii, USA: ACM Press, 2002, pp. 387–395.

[108] G. Ateniese, B. de Medeiros, and M. T. Goodrich, "TRICERT: A distributed certified E-mail scheme," in *Network and Distributed System Security Symposium (NDSS'01)*, San Diego, California, February 2001.

[109] G. Ateniese and C. Nita-Rotaru, "Stateless-recipient certified e-mail system based on verifiable encryption," in *CT-RSA*, 2002, pp. 182–199. [Online]. Available: citeseer.nj.nec.com/ateniese02statelessrecipient.html

[110] ——, "Stateless-recipient certified e-mail system based on verifiable encryption," in *RSA 2002*, Sab Jose, CA, USA, February 2002.

[111] C. Nita-Rotaru, "TURMS: A non-invasive certified email system," *http : //citeseer.nj.nec.com/*399111*.html*.

[112] B. Pfitzmann, M. Schunter, and M. Waidner, "Provably secure certified mail," IBM Research Division, Zurich, Tech. Rep. RZ 3207 (#93253), February 2000. [Online]. Available: citeseer.nj.nec.com/pfitzmann00provably.html

[113] B. Schneier and J. Riordan, "A certified e-mail protocol," in *ACSAC*, 1998, pp. 347–352. [Online]. Available: citeseer.nj.nec.com/article/schneier98certified.html

[114] J. Al-Muhtadi, R. Campbell, A. Kapadia, M. D. Mickunas, and S. Yi, "Routing through the mist: Privacy preserving communications in ubiquitous computing environments," in *Proceedings of International Conference of Distributed Computing Systems (ICDCS 2002)*, July 2002. [Online]. Available: citeseer.nj.nec.com/506969.html

[115] O. Berthold, H. Federrath, and M. Köhntopp, "Project "Anonymity and Unobservability in the Internet"," in *Workshop on Freedom and Privacy by Design / CFP2000*, 2000. [Online]. Available: citeseer.nj.nec.com/berthold00project.html

[116] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–88, February 1981.

[117] Y. Guan, X. Fu, R. Bettati, and W. Zhao, "An optimal strategy for anonymous communication protocols," in *Proceedings of the 22nd International Conference on Distributed Computing Systems.* Vienna, Austria: IEEE Computer Society, July 2002, pp. 257–267.

[118] C. Gulcu and G. Tsudik, "Mixing email with babel," in *Proceedings of the 1996 Symposium on Network and Distributed System Security (SNDSS '96).* San Diego, California: IEEE Computer Society, Feb 1996.

[119] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," in *Proceedings of the 33rd Hawaii International Conference on Systems Science*, January 2000. [Online]. Available: http://www.cs.ucl.ac.uk/staff/F.AbdulRahman/docs/hicss33.pdf

[120] K. Aberer and Z. Despotovic, "Managing trust in peer-2-peer information system," in *CIKM-01, Proceedings of the 10th International Conference on Information and Knowledge Management*, 2002, pp. 294–301. [Online]. Available: http://www-2.cs.cmu.edu/ byu/papers/p406-yu.pdf

[121] P. Yolum and M. P. Singh, "Emergent properties of referral systems," in *Proceedings of the IJCAI Workshop on Intelligent Techniques for Web Personalization (ITWP)*, Acapulco, August 2003. [Online]. Available: http://www-2.cs.cmu.edu/ byu/papers/p112-yu.pdf

[122] B. Yu and M. P. Singh, "An evidential model of distributed reputation management," in *Proceedings of First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002, pp. 294–301. [Online]. Available: http://www-2.cs.cmu.edu/ byu/papers/p406-yu.pdf

Curriculum Vitae

Saket Kaushik is a researcher and an engineer in Web technologies including Ontologies and Web Services. He has worked on several research problems in the security domain including network vulnerability analysis and attribute based access control. He received his bachelors degree in Mechanical Engineering from Indian Institute of Technology-Bombay in 1999, and his Masters degree in Information Systems from George Mason University in 2001. During his Ph.D. at George Mason University, Saket taught courses in the Software Engineering department (SWE) and also worked extensively on resolving basic computer-science related problems with Web technologies like RDF and XACML. He has several international publications to show for his effort in this domain.