# Policy Transformations for Preventing Leakage of Sensitive Information in Email Systems

Saket Kaushik[1], William Winsborough[2], Duminda Wijesekera[1], and Paul Ammann[1]

[1]Department of Information & Software Engineering, George Mason University, Fairfax, VA 22030, U.S.A,
{skaushik|dwijesek|pammann}@gmu.edu
[2]Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249-0667 USA, wwinsborough@acm.org

*In this paper we identify an undesirable side-effect 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, graph-turing tests, etc., inherently leak information, and without discontinuing their use, leakage channels cannot be closed. We formalize the capabilities of an attacker and show how she can launch guessing attacks on recipient's mail acceptance policy that utilizes leaky mechanism in an effort to avoid unwanted mail.*

*The attacker in our model guesses the contents of a recipient's private information. The recipients' use of leaky mechanisms allow the sender to verify her guess. We assume a constraint logic programming based policy language for specification and evaluation of mail acceptance criteria and present two different program transformations that can prevent guessing attacks while allowing recipients to utilize any email-control mechanism in their policies.*

**Keywords:** Application layer security, inference attacks, information leakage channels, secrecy

## 1 Introduction

Email, a widely popular communication medium, is plagued with several problems like delivery of unsolicited commercial or fraudulent messages, lack of authentication of message senders, inability to ensure integrity and secrecy of message content, *etc.* Several solutions have been proposed to counter these problems and many have been incorporated into the delivery mechanisms. However, there exists a class of problems that has not received much attention yet, which is the problem of protection of recipients' sensitive information. It is surprisingly easy to uncover information that recipients may consider sensitive, like recipient maintained *blacklist* or *whitelist*. Not only can this lead to security breaches; it can also jeopardize the defenses against unwanted messages. In this paper, we formalize this problem and a new attack technique on policy based evaluation, which is a counterpart to dictionary attacks on cryptographic protocols [3]. As a solution we also provide a policy transformation technique to prevent attacks on sensitive information.

Leakages can occur in many ways. For instance, simple *address harvesting* attacks through the Simple Mail Transfer Protocol (SMTP [15]), the default email delivery protocol, are easy to construct. 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 [13]. 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 [8].

Because email-control techniques in use at a mail server can send feedback out of band 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 [12] respond to incoming messages with a puzzle that can only 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 paper, 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 and it is the security goal of this paper 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. In section 2 we provide additional 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 [8]) 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 modeling of an attacker assumes basic capabilities of computing unfold/fold transformations [16], computing Clark completion of predicate definitions, and the ability to generate a large number of messages. Though, in the worst case analysis, the attacker need send only a $O(n)$ number of message, where $n$ is the size of the policy.

### 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 [18]) to requests for additional information (like human verification tests [12]). 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. However, it is our view that such a (non-trivial) task cannot be entrusted to end users. The only option that remains is to automatically 'strengthen' or sanitize policies susceptible to leakage; a non-trivial problem addressed here.

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.

### 1.2 Our contribution

The main contributions of this paper include what is to the best of our knowledge the first formal analysis of confidentiality problems in the context of emails, and a novel solution to protect sensitive information from attacks. In summary:

- We develop a logical formalism for expressing and solving the problem of leakage of private information due to the use of leaky mechanisms.
- We define a new attacker model with the attacker being capable of computing *Clark completion* of programs and applying *unfold/fold transformations* in addition to the ability of generating messages. We show that this is enough to uncover information considered sensitive by the message recipient.
- We describe a new type of information leakage attack on email systems due to the combination of email-control mechanisms.

- We develop two policy transformation schemes, namely, necessary and sufficient policies that, when used in tandem, can prevent the leakage of sensitive email information.

The rest of the paper is organized as follows. In section 2 we provide some motivating examples of information leakage attacks. Formal model of security policies for email are presented in section 3, followed by the attacker model in section 4. In section 5 we discuss the transformation algorithm and necessary and sufficient policy transformations that can prevent leakage of information, followed by the related work (section 6) and the conclusion (section 7).

## 2 Examples

We focus on automatic leakage of information through the email system. A simple leakage scenario is one where specially crafted messages can lead to recipients divulging private financial information to attackers. Such attack techniques are termed as 'phishing' and are beyond the scope of this paper. 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.

### 2.1 Direct disclosure

SMTP, the default email protocol, allows leakage of information, as discussed earlier. In table 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 instance, graph-turing tests [12] 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 paper, we assume that these disclosures can be prevented using policy-based control schemes for feedback control [8] and don't investigate them further.

| **Reply Code** | **Meaning** | **Confirmation provided** |
|---|---|---|
| 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 |

**Table 1:** Leakage through SMTP reply codes

### 2.2 Disclosure through leaky mechanisms

Mechanisms that provide feedback beyond the SMTP reply codes are called leaky as they can reveal information even if all SMTP feedback is prevented. For instance, bond seizure [10] is one such means. 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.

**Example 1 (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. We represent this policy informally next. A formal definition of syntax is presented later.*

| | | |
|---|---|---|
| $accept$ | $-if-$ | *some ‘allow’ rule is true and all ‘disallow’ rules are false* |
| $allow$ | $-if-$ | *sender is not blacklisted and message is bonded with value a* |
| $allow$ | $-if-$ | *for all other senders message is bonded with value b $>$ a* |
| $disallow$ | $-if-$ | *if message has an attachment with extension .scr* |

*First, assume the sender knows that an acceptance policy uses blacklists and bonds together, but doesn’t know the values a and b. The sender can send a large number of messages with different bonds and analyze seizure information to deduce a and b. If on the other hand, policies are shared, the values are already known. 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 $\in$ (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.*

## 3 Formal Model

We assume that each message is evaluated by a single evaluation engine, unlike other proposals [8]. We reconcile this design decision with existing proposals by admitting a syntax that is more general, and can be specialized according to the needs.

### 3.1 Syntax

**Definition 1 (Constraint domain).** *We use finite integer domain as the constraint domain, represented by $\mathcal{R}$, that supports standard interpretation of the symbols =, $\neq$, $\leq$ and $\geq$. We assume that non-numeric constants can be encoded in finite integer domain.*

**Definition 2 (Terms).** *Terms consist of only variables and constants. Constants are from the set $\mathcal{R}$. Tuples of terms $t_1, \ldots, t_N$ may be represented by $\overrightarrow{t}$.*

**Definition 3 (Primitive constraint).** *A primitive constraint is of the form $q(t_1, t_2)$ where q is a symbol from the set $\{=, \neq, \leq, \geq\}$ and $t_1$, $t_2$ are terms such that $t_1$ is a variable and $t_2$ is a constant. We use infix notation to represent primitive constraints.*

**Definition 4 (Constraint).** *A constraint is conjunction ($\wedge$) of primitive constraints.*

**Definition 5 (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 top level predicate symbols* allow *and* disallow $\in R_D$ *and* accept $\in R_U$. .

We treat a message as a set of facts that constrain email message headers and content to sender supplied values. For instance, `Mail From: abc@xyz` is encoded as $atrb_{\text{From}}(abc@xyz.com)$ where $atrb_{\text{From}}$ is an R$_D$ predicate. Example R$_D$ predicates include $atrb_{\text{Bond}}$ – representing attached bonds, $atrb_{\text{Attachment}}$ representing attachments, *etc.* Predicates required for defining *allow, disallow* predicates are included in R$_D$, as discussed in the definition of clauses below.

**Definition 6 (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 7 (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$_j MatchP_i$ and Q$_j MatchNotP_i$*

**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 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 5).*

**Definition 8 (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 9 (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. H atom, B, Q literals.*

**Definition 10 (CLP Program).** *A CLP Program (simply a program) is a set of clauses. For a program $\Pi$ and a predicate P, P $\propto \Pi$ if for any rule H $\leftarrow$ B$_1$,. . .,B$_n$ in $\Pi$, P = H$\theta$ or P = B$_i\theta$ (i $\in$ [1,n]) for some $\theta$.*

**Definition 11 (Message).** *A message is a set of facts*

**Definition 12 (Mail Acceptance Policy).** *A mail acceptance policy, or simply, a policy is a pair $\Pi = \langle \Pi_R, \Pi_D \rangle$ where $\Pi_R$ is a set of rules (ruleset) and $\Pi_D$ is a set of facts. The program $\Pi_R$ 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*

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

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.

### 3.2 Semantics

We reuse the three-valued semantics (with constructive negation) used in [8], which is Fages' fully abstract semantics ($T_P(I)$= $\langle T_P^+(I)$, $T_P^-(I)\rangle$) where symbols are as defined in [6], P = $\Pi \cup$ M where M is a message and $I$ = $\langle I^+$, $I^- \rangle$ in which $I^+$ and $I^-$ are disjoint sets of constrained atoms, defined next.

**Definition 13 (Constrained atom).** *A constrained atom is a pair c$|$A in which c is a solvable constraint, A is an atom and free variables occurring in c also occur as free in A. The set of all constrained atoms is denoted by $\mathcal{B}$*

**Definition 14.** *Immediate consequence function*
$T_P^+(I)$ *=*$\{$*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* $\}$
$T_P^-(I)$ *=*$\{$*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* $\}$

**Definition 15.** *Ordinal powers of $T_P$*
*$T_P \uparrow 0 = \emptyset$; $T_P \uparrow \beta = T_P(T_P \uparrow \beta - 1)$, $\beta$ is a successor ordinal; $T_P \uparrow \alpha = \bigsqcup_{\beta<\alpha} T_P \uparrow \beta$, in which $\alpha$ is a limit ordinal and $\bigsqcup_{\beta<\alpha} T_P \uparrow \beta = \langle \bigcup_{\beta<\alpha}(T_P \uparrow \beta)^+, \bigcup_{\beta<\alpha}(T_P \uparrow \beta)^- \rangle$.*

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 [8].

**Definition 16 (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$|$ $p(\overrightarrow{x})$*

Space constraints prohibit us from defining Clark completion here, so we refer the reader to Jaffar and Maher's survey on CLP [7]; later examples can help unfamiliar readers understand the concept better. We represent completion of a predicate p by p$^*$.

# 4 Attacker Model

An attacker is constrained to unlimited, but legal runs of the SMTP protocol. We make following (worst-case) assumptions:

1. Form of policies used at an email domain may be known, *e.g.*, use of blacklists, whitelists, filters, *etc.* In particular $\Pi_R$ (rule set) may be known but not $\Pi_D$ (set of facts) where contains definitions of private predicates.
2. By observing protocol runs alone an attacker cannot conclude if a message was delivered. (Recipient may silently drop messages.)
3. Recipient acts on every delivered message, like, seizes bond for unwanted message, *etc.*

## 4.1 Capabilities

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 [16]:** Given a rule $\pi_k$: H $\leftarrow$ A, B, C where A, C $\subset$ P and B $\in$ P such that for some rule $\pi_i$ and some $\theta$ such that B = head($\pi_i$)$\theta$, the attacker can transform $\pi_k$ to H $\leftarrow$ A, body($\pi_i$)$\theta$, C (here *head* and *body* functions map a rule to the atom in its head and literals in its body, respectively). Note that variables in $\pi_i$ and $\pi_k$ are renamed apart. We represent the fully unfolded form of a predicate q by q$^\omega$.

3. **Capability of fold transformation [16]:** Given a rule $\pi_k$: H ← A, B, C where A, B, C $\subset$ P such that for some rule $\pi_i$ and some $\theta$ such that B = body$(\pi_i)\theta$, the attacker can transform $\pi_k$ to H ← A, head$(\pi_i)\theta$, C
4. **Capability of message generation:** An attacker can generate any number of messages (M$_i, \ldots,$ M$_n$) of her choice.

(For additional information on unfold/fold transformation of logic programs the reader is referred to [16]).

### 4.2 Scripting an attack

Next we show how an attack can be effected using capabilities defined above. Essentially, the steps to an attack involve computing the fully unfolded form of *accept* predicate, followed by computing its Clark completion. With this computed predicate an attacker can design messages effectively, to verify her guesses. A sample attack is shown next. The unfold/fold transformation belongs to NP complexity class [2], as does the Clark completion operation. Overall, the complexity of policy attack is NP.

**Example 2.** *We provide the formal syntax of policy in example 1 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*$_{\rm bond}$ *is a leaky predicate. Suppose* $\Pi_R$ *is*

$$\begin{array}{rcl} allow(\overrightarrow{m}) & \leftarrow & \neg blacklist(Y), atrb_{\rm bond}(X), X \geq 5 \\ allow(\overrightarrow{m}) & \leftarrow & blacklist(Y), atrb_{\rm bond}(X), X \geq 10 \end{array}$$

*Using the unfolding capability, accept predicate definitions can be transformed to:*

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

*Next the attacker can compute Clark completion of accept definition:*

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

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

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

*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.*

$$Q_u(\overrightarrow{Y_u}){:}-\ Q_1(\overrightarrow{Y_1}), \ldots, \neg Q_v(\overrightarrow{Y_v}), p_1(\overrightarrow{X_{1,1}}), \ldots, p_1(\overrightarrow{X_{m_1,1}}), \neg p_1(\overrightarrow{X_{m_1+1,1}}), \ldots,$$
$$\neg p_1(\overrightarrow{X_{m_1+n_1,1}}), \ldots, p_{t'}(\overrightarrow{X_{t',1}}), \ldots, \neg p_{t'}(\overrightarrow{X_{m_{t'}+n_{t'},t'}}), c.$$

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

$$pesQ_u(\overrightarrow{Y_u}){:}\ -Q_u matchP_1(\overrightarrow{X_1}, \overrightarrow{m}), Q_u matchNotP_1(\overrightarrow{X_1}, \overrightarrow{m}).$$
$$\vdots$$
$$\text{pesQ}_u(\overrightarrow{Y_u}){:}\ -Q_u matchP_{t'}(\overrightarrow{X_{t'}}, \overrightarrow{m}), Q_u matchNotP_{t'}(\overrightarrow{X_{t'}}, \overrightarrow{m}).$$

$$Q_u matchP_k(\overrightarrow{X_{m_k+j}}, \overrightarrow{m}){:}\ -pesQ_1(\overrightarrow{Y_1}), \ldots, \neg optQ_v(\overrightarrow{Y_v}), Q_u matchP_1(\overrightarrow{X_{1,1}}, \overrightarrow{m}), \ldots,$$
$$Q_u matchP_1(\overrightarrow{X_{m_1,1}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{1,k}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{m_k,k}}, \overrightarrow{m}),$$
$$Q_u matchNotP_k(\overrightarrow{X_{m_k+1,k}}, \overrightarrow{m}), \ldots, Q_u matchNotP_k(\overrightarrow{X_{m_k+(j-1),k}}, \overrightarrow{m}),$$
$$Q_u matchNotP_k(\overrightarrow{X_{m_k+(j+1),k}}, \overrightarrow{m}), \ldots, Q_u matchNotP_k(\overrightarrow{X_{m_k+n_k,k}}, \overrightarrow{m}), \ldots,$$
$$Q_u matchP_{t'}(\overrightarrow{X_{1,t'}}, \overrightarrow{m}), \ldots, Q_u matchNotP_{t'}(\overrightarrow{X_{m_{t'}+n_{t'},t'}}, \overrightarrow{m}), \overrightarrow{X_{i,k'}} \neq \overrightarrow{X_{m_{k'}+j,k'}}, c.$$
$$Q_u matchNotP_k(\overrightarrow{X_i}, \overrightarrow{m}){:}\ -pesQ_1(\overrightarrow{Y_1}), \ldots, \neg optQ_v(\overrightarrow{Y_v}), Q_u matchP_1(\overrightarrow{X_{1,1}}, \overrightarrow{m}), \ldots,$$
$$Q_u matchP_1(\overrightarrow{X_{m_a,1}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{1,k}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{i-1,k}}, \overrightarrow{m}),$$
$$Q_u matchP_k(\overrightarrow{X_{i+1,k}}, \overrightarrow{m}), \ldots, Q_u matchP_k(\overrightarrow{X_{m_k,k}}, \overrightarrow{m}), Q_u matchNotP_k(\overrightarrow{X_{m_k+1,k}}, \overrightarrow{m}), \ldots,$$
$$Q_u matchNotP_k(\overrightarrow{X_{m_k+n_k,k}}, \overrightarrow{m}), \ldots, Q_u matchP_{t'}(\overrightarrow{X_{1,t'}}, \overrightarrow{m}), \ldots,$$
$$Q_u matchNotP_{t'}(\overrightarrow{X_{m_{t'}+n_{t'},t'}}, \overrightarrow{m}), \overrightarrow{X_{i,k'}} \neq \overrightarrow{X_{m_{k'}+j,k'}}, c.$$
$$optQ_u(\overrightarrow{Y_u}){:}\ -optQ_1(\overrightarrow{Y_1}), \ldots, \neg pesQ_v(\overrightarrow{Y_v}), \overrightarrow{X_{i,k'}} \neq \overrightarrow{X_{m_{k'}+j,k'}}, c.$$

**Figure 1:** Transformation algorithm

## 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, hence 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.

### 5.1 Transformation algorithm

Transformation algorithm is discussed next. Since 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 1 and consists of two transformations for each rule containing sensitive predicates and is described in detail next.

Figure 1 begins with a general Horn clause representation of rules in $\Pi_R$ with meta-variables Q$_u$, Q$_v$ and p$_k$ and $\overrightarrow{m}$ is the tuple of all variables used in $\Pi_R$. $Q_u(\overrightarrow{y})$ 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, the stratification restriction. Also, each $p_k$ literal need not appear in the body of every $Q_u$ clause, *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.*, $pesQ_u$ and $optQ_u$, where $pesQ_u$ is the ‘pessimistic’ version of $Q_u$, independent of the definition of any private predicate used in the definition of $Q_u$, and $optQ_u$ is the ‘optimistic’ version of $Q_u$ predicate, which holds for ‘some’ definition of private predicates. More precisely, $optQ_u$ 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 $pesQ_u$ 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 $pesQ(\overrightarrow{x})$ whenever it is used positively, and $optQ(\overrightarrow{x})$ 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 3 we present a concrete example of this transformation.

**Example 3.** *Pessimistic and optimistic transformations.*
*Consider the $\Pi_R$ 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 $\Pi_D$:*

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

*The optimistic and pessimistic forms of the predicate trusted in $\Pi_{suf}$ 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 trustedMatchNotBlacklist predicate due* *to space constraints:*

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

### 5.1.1 Necessary Policy

Intuitively, the necessary policy, $\Pi_{nec}$, 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 4 illustrates the basic idea:

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

**Example 4.** *[Illustration of necessary policy] Consider a ruleset $\Pi_R$ 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, $p \in \mathcal{P}$*

$$allow(\overrightarrow{msg}) \quad \leftarrow \quad B_1, p(X) \tag{1}$$

$$allow(\overrightarrow{msg}) \quad \leftarrow \quad B_2, \neg p(X) \tag{2}$$

*Applying the necessary transformation we get:*

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

*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 1,2*

### 5.1.2 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 [14]. 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 5 (Illustration of sufficient policy).** *Consider the ruleset given by rules 1 and 2. The sufficient transformation of rules yields the following ruleset*

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

*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 1 and 2, with private predicate stripped off from the rules.*

### 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 1.** *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.* $\Pi_{suf}$*) where* $\Pi_{nec}$ *(resp.* $\Pi_{suf}$*) is necessary (resp. sufficient) transformation of* $\Pi_R$.

**Corollary 2.** *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.* $\Pi^{*}_{suf}$*).*

### 5.3 Semantic Properties

To show how evaluation of $\Pi_{nec}$ and $\Pi_{suf}$ instead of $\Pi_R$ prevents sensitive leakages, we need to show some semantic properties of the transformed rulesets. However, space constraints don't allow us to go into the full details of our claims, the theorems and their proofs. Hence, we briefly describe the results informally and refer the interested reader to a technical report [9] with complete results. However, here we state our main theorem without its proof.

We use the following notations. The program corresponding to the original policy is represented by P, where P = $\Pi_R \cup \Pi_D \cup M$, in which $M$ is the message being evaluated, $\Pi_D$ is the set of private facts and $\Pi_R$ is a ruleset. The sufficient transformation yields a set of rules represented by $\Pi_{suf}$, whereas the necessary ruleset is represented by $\Pi_{nec}$. Assuming $\Pi_D$ contains only facts constructed from private predicates, we denote the program corresponding to $\Pi_{suf}$ by $P_S$, where $P_S$ = $\Pi_{suf} \cup M$ and the program corresponding to $\Pi_{nec}$ by $P_N$, where $P_N$ = $\Pi_{nec} \cup M$. Both these programs are independent of the definitions of the sensitive predicates.

The main theorem involves a general relation between satisfaction of 'optimistic' and 'pessimistic' versions of any literal and the satisfaction of the literal itself. 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, $\Pi_D$), such that the original predicate is satisfied.

**Theorem 3.** *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 $\Pi_D$ 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.*

With the help of the theorem above, semantic closeness of transformations to the original policy can be shown in a straightforward manner. That is, transformed policies are closest, semantically, to the original policy compared to any other policy that protects recipient's private information. Based on semantic closeness, email policies can be partially ordered (additional details can be found in [9]) and it can be shown that the necessary policy is the least upper bound for all policies that protect recipient's sensitive information, while the sufficient policy the greatest lower bound.

**Protection against attacks** Under the assumed capabilities of the attacker, the above results enable us to prove certain results regarding the protection offered by the transformations. We summarize when an attacker can gain sensitive information and when the information is protected, next. (For proofs see [9]).

- The attacker can gain knowledge of sensitive information (*i.e.*, portions of $\Pi_D$) if she knows the original ruleset (*i.e.*, $\Pi_R$) and messages (*i.e.*, M$_i$) are evaluated by the original policy (*i.e.*, the program $\Pi_R \cup \Pi_D \cup M_i$).
- Attacker's knowledge of necessary and sufficient policy and evaluation of messages by these policies does not lead to leakage of sensitive information.
- Attacker's knowledge of original policy and evaluation of messages by sufficient and necessary policies will not lead to leakage of sensitive information.

## 6 Related Work

Cryptanalysis of private-key cryptosystems through statistical attacks, like correlation attacks [11], aim to determine the statistical relationship between outputs and inputs of cryptographic transformations. Zhang, Tavares *et al.* [19] describe a zero information leakage between the change of output(s) and prescribed change patterns in the inputs for protecting against correlation attacks. Our approach resembles this information theoretic model of protection against information leakage, however, we describe how correlation-like attacks can be mounted against sets of Horn clauses and present a transformation that can prevent against such attacks.

Our transformation procedure resembles the predicate elimination strategy, a complete resolution proof strategy for multi-predicate formulas, proposed by Reiter [14]. Essentially, this strategy involves rewriting the theory with a predicate P 'resolved away'. Subsequently, a set of unsatisfiable P-independent clauses can be derived if the original set of clauses were unsatisfiable. In our approach, we propose a strategy for 'resolving away' the private predicates in a given set of rules. However, our aim here is not to detect unsatisfiability. Instead, we construct new clauses that do not leak any discernible information to guessing attacks.

The third closely related work is of Delaune and Jacquemard [3], who give a theory of dictionary attacks against cryptographic protocols. In their work, they claim that if the set of possible values of the input is finite (and small), then a dictionary attack (guessing attack) is only PTIME complex. They go on to give a theory of dictionary attack by extending the classic Dolev-Yao intruder model for statistical inferences. In our work, we adopt their attack model, and even though we require the attacker to be able to handle a greater degree of computational complexity, the basis of launching attacks remains the same.

Relational databases have mature techniques for both access control and inference control. Access control protects direct access to sensitive information. In our case, we assume that this is possible by policy specification and enforcement. Inference control has been extensively studied in statistical databases and census data [4, 17, 1]. These approaches can be classified into *restriction-based*, or restricting queries, or *perturbation-based*, *i.e.*, addition of random noises to source data. Our approach is closer to the restriction based techniques.

In restriction based inference control schemes, one of the concerns is of an attacker deriving protected information through aggregation of separate queries. In other words, the protected information cannot be queried directly, but deducible from the results of other queries. In the email domain, a query can be replaced by a message, and the result of a query by a yes or no decision (*i.e.*, accept or a reject). Even with a boolean response, attackers can deduce relevant information. This is the reason why we claim that inference attacks are easier to construct. Similar to their response, we transform the evaluation policies, and thus reduce attacker's capabilities to run some queries.

In summary, we have applied a well-studied problem to the context of email messages and showed that important information can be lost due to the current email delivery protocols and deployed mechanisms. Solutions applied to other domains are not directly applicable to our domain, and therefore we provide a custom solution based on program transformations, using ideas developed by researchers who have studied similar problems in other domains.

## 7 Conclusion

In this paper we have 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.

As opposed to the classical Dolev-Yao attacker [5], we define a new attacker model and a new attack technique. 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 such that sensitive information is leaked. As a solution, we provide an algorithmic transformation which can sanitize the combination of email-control mechanisms, so that the leakage is plugged. We also show that the transformed policies that we generate are ‘closest’ semantically to the original policy.

## References

[1] N. R. Adam and J. C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.

[2] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[3] S. Delaune and F. Jacquemard. A theory of dictionary attacks and its complexity. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 2–15, 2004.

[4] D. E. Denning and J. Schlrer. Inference control for statistical databases. *IEEE Computer*, 16(7):69–82, 1983.

[5] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transaction on Information Theory*, 29:198–208, 1983.

[6] F. Fages. Constructive negation by pruning. *Journal of Logic Programming*, 32/2, 1997.

[7] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[8] 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)*, pages 73–82, Fairfax, VA, November 2005.

[9] S. Kaushik, W. Winsborough, D. Wijesekera, and P. Ammann. Policy transformation for preventing leakage of sensitive information in email systems. Technical Report ISE-TR-06-05, ISE Dept, George Mason University, Fairfax, VA, May 2006.

[10] T. Loder, M. V. Alstyne, and R. Wash. An economic solution to the spam problem. In *ACM E-Commerce*, 2004.

[11] W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[12] M. Naor. Verification of a human in the loop or identification via the turing test. http://www.wisdom.weizmann.ac.il/ñaor/ PAPERS/human_abs.html, 1996.

[13] S. Petry. Port 25: The gaping hole in the firewall. In *Proceedings of ACSAC'02 Annual Computer Security Applications Conference*, Dec 2002.

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

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

[16] H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-A. Tarnlund, editor, *Proceedings of the Second International Conference on Logic Programming*, pages 127–138, Uppsala, 1984.

[17] L. Willenborg and T. de Waal. *Statistical disclosure control in practice*. Springer Verlag, New York, 1996.

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

[19] M. Zhang, S. Tavares, and L. Campbell. Information leakage of boolean functions and its relationship to other cryptographic criteria. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security (CCS'94)*, pages 156–165, Fairfax, 1994.