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. This is 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 its defence against unwanted mail.

As opposed to the classical Dolev-Yao attacker and its extensions, attacker in our model guesses the contents of a recipient's private information. The 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, Dolev Yao

I. 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, but 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 [4]. 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 [21]), the default email delivery protocol, are easy to construct. In this attack, a malicious sender attempts delivery to a preconstructed list of possible recipients, and recipient mail server replies help her to identify which ones are *bona fide* account holders [18]. 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 [9], [10].

Controlling SMTP feedback to senders is not enough to protect recipient's private data. For instance, email-control techniques like monetary bonds [14], graph-turing tests for human initiation [17], *etc.*, provide feedback to senders from outside the SMTP transmission channel. Clearly, signals through monetary transfer due to bond seizure can't be prevented by stopping SMTP feedback. This signal informs the sender that the sent message was able to overcome recipient's bayesian filters, if they are being used by the recipient, in addition to confirming recipient's email address. 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, by the virtue of its use, 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 II we describe how leaky mechanisms provide message acceptance confirmations to the sender. 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], [9], [10]) compound the problem. Armed with the knowledge of recipient policies, an attacker can simply send a large volume of messages and observe the behavior of the feedback channel in order to discern relevant information.

Our modeling of an attacker assumes basic capabilities of computing unfold/fold transformations [22], computing Clark completion of predicate definitions, and the ability to generate a large number of messages. Though, in the worst case analysis, this attacker need not generate a large number of messages to learn parts of a recipient's private information. With this attacker in mind, and 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.

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 [19], 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. In the section VII we survey some of the disclosure solutions and argue why they are different from our domain.

A. Our Contribution

The main contributions of this paper include, to the best of our knowledge, the first formal analysis of well-studied confidentiality problem in the context of emails, and a novel solution to protect sensitive information from attacks. We summarize our contributions as follows:

- 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. With these capabilities, and the usual assumptions about distributed email communications, this attacker can unravel message recipients' protected information.
- 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 II we provide some motivating examples of information leakage attacks. We follow this with an informal and intuitive description of how we plan to mitigate leakage attacks in section III. This is followed by the formal model in section IV and the attacker model in section V. In section VI we discuss the transformation algorithm and necessary and sufficient policy transformations that can

prevent leakage of information, followed by the related work (section VII) and the conclusion (section VIII).

II. Examples

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. We focus on automatic leakage of information through the email system. Several types of information may be regarded as lucrative by different classes of message senders. For instance, knowledge of a large set of valid email addresses is of prime importance to bulk emailers. As would be the efficacy of filtering rules of bayesian filters, now omnipresent in every recipient email system. 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.

A. Direct disclosure

SMTP, the default email protocol, allows leakage of information, as discussed earlier. In table I 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 to the SMTP protocol, email-control schemes and protocols layered on top of the SMTP protocol can also result in leakage of information. For instance, graph-turing tests [17] is one such protocol that can be employed by recipients for proof of human initiation. In this scheme, recipients generate a challenge for incoming messages, and only on receiving a successful response is the message delivered to recipient's mailbox. However, issuing a challenge confirms that the recipient address is a valid email address, or it successfully dodged bayesian filters employed by the recipient.

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

TABLE I: Leakage through SMTP reply codes

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 [9], [10] and don't investigate them further.

B. 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 which or what type of messages can contain HTTP content. Though URLs and images don't qualify as a leaky mechanism (since they are not email-control mechanisms), similar leakages are possible through other mechanisms. For instance, 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:** Bulk senders may simply wish to know if a recipient address is valid or not and can use leaky mechanisms, SMTP feedback or well-crafted messages for this purpose. Leaky and sensitive mechanisms need not be combined for this leakage to occur.
- **Leakage of sensitive information:** In this case, the sender knows that the recipient address is valid and wishes to know additional information about a recipient, using the fact that the recipient employs leaky and sensitive mechanisms in combination.

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

Example 1 (Leakage through monetary bonds). *Consider a simple recipient policy that states that messages from blacklisted email domains (or senders) will be accepted if a minimum bond of value $ b is present; for all other messages a bond of minimum value $ a (a < b) is required. Such a scenario is easily foreseen as monetary signalling techniques ([11], [14], [16], [23], etc.) have recently been applied (in various capacities) in real email networks. We represent this policy simply as (a formal definition of syntax follows later):*

$$\begin{aligned} accept\ message \quad &-if- \quad sender\ is\ not\ blacklisted\ and\ message\ is\ bonded\ with\ value\ a \\ accept\ message \quad &-if- \quad sender\ is\ blacklisted\ and\ message\ is\ bonded\ with\ value\ b > \ a \end{aligned}$$

Suppose a sender knows that the recipient is using a policy based on bond values and blacklists. Further, since bond seizure provides confirmation if a message was read or not, 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 the values of a and b known, the sender can easily verify if an email address is in the recipient's blacklist or not. The situation is further simplified, if the sender is given these values as a part of policy sharing. Such an attack can be accomplished with as little as only two email messages: the sender sends two identical messages with only bond values different: one bonded with value $ c, c ∈ (a,b) and other bonded with value $ d, d > b. Also, we assume that the targeted recipient seizes bonds for all commercial messages delivered to his mailbox. If both the bonds are seized, the sender knows that he or she is not on the blacklist, otherwise if only the second bond is seized will prove that the sender is on the blacklist.

III. Overview of our approach

Controlling unwanted messages through the use of various control techniques simultaneously and at various stages of email delivery has been studied and proposed recently ([2], [12], [13] among others). Policy based control has also been proposed recently that allows flexible control of email delivery mechanism, while allowing the basic protocol to treat each email-control technique uniformly. Thus, turing tests, and other human initiation checks [17] can be integrated within the SMTP framework, through policy feedback or policy sharing. However, this opens up the possibility of any (and all) email control requirements to be communicated upstream. This capability gives policies an important role in email delivery, with the potential of drastically altering the way unwanted messages are handled, *i.e.*, unwanted messages may not allowed to originate at the sender domain if they don't satisfy downstream policies. However, enforcement of this scheme requires explicit policy sharing with upstream agents. This readily enables attackers to launch and verify guessing attacks. Even without explicit sharing, guesses can be verified through leaky mechanisms, though the complexity of such attacks is more than the policy sharing case.

Even though communication or enforcement of policies that combine leaky and sensitive mechanisms is vulnerable to attacks, and therefore, requires policies to be 'strong enough' to withstand leakage analysis, the responsibility of enforcing 'strong' policies cannot be entrusted to the end users. At best, the end users are expected to answer simple questions that help the system to construct a message acceptance policy on their behalf. The only option that remains is to automatically strengthen or sanitize a policy that can leak sensitive information. This is not a trivial problem and in the sections to follow, we show how to solve it. However, before we outline our solution, it is worth noting that 'strengthening' action may not be required in every SMTP session. Based on mutual trust, and history of previous interactions, email domains may distinguish between different transmitting domains and decide on whether to use policy sanitization actions or not. (For additional information on such 'service level' decisions the reader is referred to [8], [10].)

Our first step in policy sanitization is to distinguish leaky mechanisms and sensitive information from the rest of the recipient policy. This is required as we need to focus on the protected resource and the means through which leakage occurs. The distinction is made in the syntax of policies and details are presented in section IV. Next, we transform the original recipient policy into a zero information leakage policy against a correlation attack. For example, consider again the policy in example 1. Assuming the policy only evaluates the sender's membership in blacklist and monetary value of the bond, clearly, a message must at least have a bond value of $ a to be accepted and will always be accepted if the bond value is at least $ b. We encode this information in two new policies: the *necessary policy* that states that a message must have a bond value of $ a and the *sufficient policy* that states that message need only have a bond value of $ b to be acceptable. Note that in both the policies all references to sensitive mechanism, *i.e.*, blacklist, have been removed. Evaluating the sufficient policy for every

message, clearly, does not yield information regarding the contents of recipient's blacklist. To establish minimum threshold, the necessary policy can be sent across the network without the risk of leakage analysis, while message evaluations being performed with sufficient policy in the worst case. Policy transformations are detailed in § VI and we prove that transformations achieve required security goals, while being semantically 'closest' to original policy, in § VI-C,VI-D.

IV. Formal Model

A formal model for a policy based decision on email acceptance was presented earlier [8], [10], [9]. Here we do not go into the details of the earlier models, but discuss a more general 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 [10]. As our syntax is more general, it can be specialized to represent any of the policies suggested earlier, or their composition.

A. 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 $\geq$.*

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 predefined 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 three top level predicate symbols* accept, allow *and* disallow $\in R_D$.

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 VI).*
- 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 VI).*

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.*

Definition 10 (CLP Program). *A CLP Program (simply a program) is a set of clauses. For a program P and a predicate Q, Q $\propto$ P if for any rule H $\leftarrow$ B_1,...,B_n in P, Q = Hθ or Q = $B_i\theta$ (i $\in$ [1,n]) for some θ.*

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

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.

We also assume that non-numeric constants can be encoded in finite integer domain.

Definition 12 (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*

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

B. Semantics

We reuse the three-valued semantics (with constructive negation) used in [10], which is Fages' fully abstract semantics ($T_P(I)$= $\langle T_P^+(I), T_P^-(I)\rangle$) where symbols are as defined in [7], 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
$T_P \uparrow 0 = \emptyset$; $T_P \uparrow \beta = T_P(T_P \uparrow \beta - 1)$, β *is a successor ordinal;* $T_P \uparrow \alpha = \bigsqcup_{\beta<\alpha} T_P \uparrow \beta$, *in which* α *is a limit ordinal and* $\bigsqcup_{\beta<\alpha} T_P \uparrow \beta = \langle \bigcup_{\beta<\alpha}(T_P \uparrow \beta)^+, \bigcup_{\beta<\alpha}(T_P \uparrow \beta)^-\rangle$.

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 [10]. Finally we define the extension and Clark completion of a predicate as follows

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

Definition 17 (Clark completion). *Given a ruleset* Π*, each predicate p,* $p \propto \Pi$ *such that for some rule* $\pi \in \Pi$ $p(\overrightarrow{x})$ *= head(*π*), 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

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

where y_i *is the set of variables in* B_i *except for variables in* $\overrightarrow{x_i}$*. If p* $\neq$ *head(*π*) 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^*

V. Attacker Model

Next we define the attacker's capabilities and model leakage of 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 plays 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) Attacker may know the form of policies being used at a particular email domain, *i.e.*, 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) where contains definitions of private predicates.
2) From protocol runs an attacker cannot discern if a message reached the intended recipient's mailbox, *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) Each message that reaches a recipient's mailbox is read by the recipient and actions like bond seizure, reporting to reputation services, *etc.*, are taken.

A. Capabilities

Given a set of rules $\Pi = \{\pi_1, \ldots, \pi_n\}$, and the set Q =$\{q \mid q \propto \Pi\}$, an attacker has following capabilities:

1) **Capability of computing Clark completion:** For all $q \in Q$, the attacker can compute q^*, q's Clark completion with respect to $\mathcal{P}$.
2) **Capability of unfold transformation [22]:** Given a rule π_k: H $\leftarrow$ A, B, C where A, C $\subset$ Q and B $\in$ Q such that for some rule π_i and some θ such that B = head$(\pi_i)\theta$, the attacker can transform π_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).
3) **Capability of fold transformation [22]:** Given a rule π_k: H $\leftarrow$ A, B, C where A, B, C $\subset$ Q such that for some rule π_i and some θ such that B = body$(\pi_i)\theta$, the attacker can transform π_k to H $\leftarrow$ 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 [22].) For modeling attacks on cryptographic protocols, the classical Dolev-Yao (DY) attacker [6] was used. DY attacker was able to evaluate the secrecy of data during transmission. As opposed to this attacker, the email policy attacker is capable of uncovering subset of extension of private predicate(s). The security goal of this paper is to prevent an attacker from gaining this knowledge. Computing Clark completion of program definitions, unfold/fold transformations and sending messages are the assumed capabilities *vis-a-vis* DY attacker capabilities like encryption, decryption, term concatenation, term splitting, *etc.*

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 from an analysis of 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$' \in R_A$. A precise logical definition of this attack is being worked on.

B. Scripting an attack

An attacker can generate requisite messages using her capabilities in the following way. 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$^\omega$). This operation yields accept$^\omega$, the fully unfolded form of accept with respect to $\mathcal{P}$. In the next step the attacker constructs the Clark completion of accept$^\omega$ to yield accept$^{\omega *}$. 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 [3], as does the Clark completion operation. Overall, the complexity of policy attack is NP.

Example 2. *We encode a simple attack as discussed in section II. Here, blacklist is the private predicate, whose definition*

(or extension) is hidden from the attacker and bond is the leaky predicate. Suppose Π_R *is*

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

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

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

Next the attacker can compute Clark completion of accept definition:

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

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

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

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 ∈ (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.*

VI. 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. The intuition behind sufficient policy is that it 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.

A. 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 Π_R with meta-variables Q_u, Q_v and p_k and $\overrightarrow{msg}$ 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, 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.

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

For each clause in Π_R as shown above, add create following clauses, for each k and u, if not already present:

$$pesQ_u(\overrightarrow{Y_u})\text{: }-Q_umatchP_1(\overrightarrow{X_1},\overrightarrow{msg}),Q_umatchNotP_1(\overrightarrow{X_1},\overrightarrow{msg}).$$
$$\vdots$$
$$p\text{esQ}_u(\overrightarrow{Y_u})\text{: }-Q_umatchP_{t'}(\overrightarrow{X_{t'}},\overrightarrow{msg}),Q_umatchNotP_{t'}(\overrightarrow{X_{t'}},\overrightarrow{msg}).$$

$$Q_umatchP_k(\overrightarrow{X_{m_k+j}},\overrightarrow{msg})\text{: }-pesQ_1(\overrightarrow{Y_1}),\ldots,\neg optQ_v(\overrightarrow{Y_v}),Q_umatchP_1(\overrightarrow{X_{1,1}},\overrightarrow{msg}),\ldots,Q_umatchP_1(\overrightarrow{X_{m_1,1}},\overrightarrow{msg}),$$
$$\ldots,Q_umatchP_k(\overrightarrow{X_{1,k}},\overrightarrow{msg}),\ldots,Q_umatchP_k(\overrightarrow{X_{m_k,k}},\overrightarrow{msg}),Q_umatchNotP_k(\overrightarrow{X_{m_k+1,k}},\overrightarrow{msg}),\ldots,$$
$$Q_umatchNotP_k(\overrightarrow{X_{m_k+(j-1),k}},\overrightarrow{msg}),Q_umatchNotP_k(\overrightarrow{X_{m_k+(j+1),k}},\overrightarrow{msg}),\ldots,Q_umatchNotP_k(\overrightarrow{X_{m_k+n_k,k}},\overrightarrow{msg}),$$
$$\ldots,Q_umatchP_{t'}(\overrightarrow{X_{1,t'}},\overrightarrow{msg}),\ldots,Q_umatchNotP_{t'}(\overrightarrow{X_{m_{t'}+n_{t'},t'}},\overrightarrow{msg}),$$
$$\overrightarrow{X_{i,k'}}\neq\overrightarrow{X_{m_{k'}+j,k'}},\ i\in[1,m_{k'}],\ j\in[1,n_{k'}],k'\in[1,t'],c.$$

$$Q_umatchNotP_k(\overrightarrow{X_i},\overrightarrow{msg})\text{: }-pesQ_1(\overrightarrow{Y_1}),\ldots,\neg optQ_v(\overrightarrow{Y_v}),Q_umatchP_1(\overrightarrow{X_{1,1}},\overrightarrow{msg}),\ldots,Q_umatchP_1(\overrightarrow{X_{m_a,1}},\overrightarrow{msg}),$$
$$\ldots,Q_umatchP_k(\overrightarrow{X_{1,k}},\overrightarrow{msg}),\ldots,Q_umatchP_k(\overrightarrow{X_{i-1,k}},\overrightarrow{msg}),Q_umatchP_k(\overrightarrow{X_{i+1,k}},\overrightarrow{msg}),\ldots,$$
$$Q_umatchP_k(\overrightarrow{X_{m_k,k}},\overrightarrow{msg}),Q_umatchNotP_k(\overrightarrow{X_{m_k+1,k}},\overrightarrow{msg}),\ldots,Q_umatchNotP_k(\overrightarrow{X_{m_k+n_k,k}},\overrightarrow{msg}),\ldots,$$
$$Q_umatchP_{t'}(\overrightarrow{X_{1,t'}},\overrightarrow{msg}),\ldots,Q_umatchNotP_{t'}(\overrightarrow{X_{m_{t'}+n_{t'},t'}},\overrightarrow{msg}),$$
$$\overrightarrow{X_{i,k'}}\neq\overrightarrow{X_{m_{k'}+j,k'}},\ i\in[1,m_{k'}],\ j\in[1,n_{k'}],k'\in[1,t'],c.$$

$$optQ_u(\overrightarrow{Y_u})\text{: }-optQ_1(\overrightarrow{Y_1}),\ldots,\neg pesQ_v(\overrightarrow{Y_v}),\overrightarrow{X_{i,k'}}\neq\overrightarrow{X_{m_{k'}+j,k'}},\ i\in[1,m_{k'}],\ j\in[1,n_{k'}],k'\in[1,t'],c.$$

Fig. 1: Transformation algorithm

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 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:

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

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 trustedMatchNotBlacklist predicate

due to space constraints:

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

1) 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 4 illustrates the basic idea:

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

Example 4. *[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, $p \in \mathcal{P}$*

$$allow(\overrightarrow{msg}) \leftarrow B_1, p(X) \quad (1)$$
$$allow(\overrightarrow{msg}) \leftarrow B_2, \neg p(X) \quad (2)$$

Applying the necessary transformation we get:

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

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 1,2

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 [20]. The following top-level predicate accept (accept$_{suf}$ for clarity) achieves this aim:

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

Example 5 (Illustration of sufficient policy). *Consider the ruleset given by rules 1 and 2. The sufficient transformation of rules yields the following ruleset*

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

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

B. 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 VI.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. Π_{suf}) where Π_{nec} (resp. Π_{suf}) is necessary (resp. sufficient) transformation of Π_R.* □

Corollary VI.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. Π^{*}_{suf}).* □

C. 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 VI.3. 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}.

a) '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 VI.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 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 sketch: See appendix

b) 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. In other words, we wish to show that the transformation algorithm generates 'correct' necessary and sufficient policies. Informally, just as 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.

Corollary VI.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, $c\,|accept(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ if $c\,|accept_{suf}(\overrightarrow{m}) \in T^{+}_{P_S} \uparrow \omega$.*

Proof sketch: 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 VI.3 that $c \mid pesAllow(\overrightarrow{m}) \in T^{+}_{P_S} \uparrow \omega$ if $c \mid allow(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ for all definitions of private predicates. Also, $c \mid optDisallow(\overrightarrow{m}) \in T^{-}_{P_S} \uparrow \omega$ if and only if $c \mid disAllow(\overrightarrow{m}) \in T^{-}_{P} \uparrow \omega$ for some definition, therefore, $c \mid accept(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ if $c \mid accept_{suf}(\overrightarrow{m}) \in T^{+}_{P_S} \uparrow \omega$ □

Corollary VI.5. *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, $c \mid accept_{suf}(\overrightarrow{m}) \in T^{+}_{P_S} \uparrow \omega$ if $c \mid accept(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ for all possible definitions of predicates p_k, $k \in$ [1, t].*

Proof sketch: Follows from theorem VI.3 and the definition of the predicate accept_{suf} □

Corollary VI.6. *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, $c \mid accept_{nec}(\overrightarrow{m}) \in T^{+}_{P_N} \uparrow \omega$ if $c \mid accept(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$.*

Proof sketch: 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 VI.3 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 $c \mid accept(\overrightarrow{m}) \in T^{+}_{P} \uparrow \omega$ □

c) 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 18 (Email Policy Dominance Relation). *Relation $>_\Pi$ 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 $\langle \Pi_X, \Pi_Y \rangle \in >_\Pi$ 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 $\langle \Pi_X, \Pi_Y \rangle \in >_\Pi$ by $\Pi_X >_\Pi \Pi_Y$.*

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:

- $x >_\Pi x$
- $x >_\Pi y \wedge y >_\Pi x \rightarrow x = y$
- $x >_\Pi y \wedge y >_\Pi z \rightarrow x >_\Pi z$

□

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

Proof: Follows from Corollary VI.5 □

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

Proof: Follows from Corollary VI.6 □

Theorem VI.7. *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_{suf}, the sufficient transformation of Π_R, is the least upper bound, under the policy dominance relation $>_\Pi$, that entails $\Pi_R \cup \Pi_D$, for all possible definitions of p_k, $k \in$ [1, t].*

Proof sketch: From proposition 2 $\Pi_{suf} >_\Pi \Pi_R \cup \Pi_D$ and from corollary VI.5 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 VI.5 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 VI.8. *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 $>_\Pi$, that is entailed by $\Pi_R \cup \Pi_D$, for some definition of predicates p_k, k ∈ [1, t].*
Proof sketch: The proof follows from theorem VI.3 and corollary VI.6 following similar arguments as in theorem VI.7 □

D. 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 the security goal that is addressed in this paper. In other words, we wish to show that the transformed policies are 'safe' and they provide the necessary protection against the given attacker model.

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 knows or can compute $\Pi^{\omega*}_R$ and generate verifiers, *i.e.*, 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 – 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 $\Pi^{\omega*}_R$ 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 original policy is used to evaluate messages, whereas in the other two cases the transformed policies are evaluated. We give informal proofs for the following theorems.

Theorem VI.9. *It is possible to verify a guess 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 Sketch: 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 $\Pi^{\omega*}_R$, she can generate a sequence of messages to verify her guess. □

Theorem VI.10. *If the attacker knows $\Pi^{\omega*}_{suf}$ (resp. $\Pi^{\omega*}_{nec}$) 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 ∈ ext(p), where p ∈ $\mathcal{P}$ and ext(p) is its extension*

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

Theorem VI.11. *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 ∈ ext(p), where p ∈ $\mathcal{P}$ and ext(p) is its extension*

Proof Sketch: With Π_R the attacker can construct rules that define a predicate p′ ($\in R_A$) such that ext(p′) ⊆ ext(p). However, these rules require verifiers generated from evaluation of the $\Pi_R \cup \Pi_D \cup M$. Since p $\not\propto \Pi_{suf} \cup M$ (and 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) g ∈ ext(p)
2) g ∉ ext(p)

Hence, the attacker does not get back the required verifiers to verify her guess. □

VII. Related Work

Cryptanalysis of private-key cryptosystems through statistical attacks, like correlation attacks [15], aim to determine the statistical relationship between outputs and inputs of cryptographic transformations. Zhang, Tavares *et al.* [25] 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 [20]. 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 [4], 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 [5], [24], [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.

VIII. Conclusion and Future Work

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, we define a new attacker model and an 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.

Here, we are not so concerned with feedback obtained from out of band channels, like recipient informing the sender through a telephone conversation. There is little we can hope to do about such signals, and their bandwidth is typically low. 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 [8], 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.

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

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

[4] 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.

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

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

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

[8] 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.

[9] S. Kaushik, D. Wijesekera, W. Winsborough, and P. 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)*, pages 31–45, Barcelona, Spain, October 2005.

[10] 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.

[11] 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*, pages 2006–215, New York, 2003. ACM Press.

[12] B. Leiba and N. Borenstein. A multifaceted approach to spam reduction. In *CEAS 2004: First Conference on Email and Anti-Spam*, Oakland, July 2004.

[13] 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.

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

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

[16] Microsoft's Penny Black Project. http://research.microsoft.com/research/sv/PennyBlack/.

[17] 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.

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

[19] C. Pfleeger and S. Pfleeger. *Security in Computing*. Prentice Hall, third edition, 2003.

[20] 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.

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

[22] 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.

[23] B. Templeton. E-Stamps. http://www.templetons.com/brad/spam/ estamps.html.

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

[25] 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.

Appendix

Proof Sketch of theorem VI.3:

In all the three programs, a literal $Q(\overrightarrow{y})$ can be proven if $c\,|Q(\overrightarrow{y}) \in T_P^+ \uparrow \omega$. Since 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 $c \mid p_k(\overrightarrow{x}) \in T_P^- \uparrow \alpha$, $c \mid Q_u\ matchP_k(\overrightarrow{x}, \overrightarrow{msg}) \in T_{P_S}^+ \uparrow \alpha$

entails $c \mid Q_u(\overrightarrow{x}) \in T_P^+\uparrow\beta$ for some β. Similarly, it states that given $c \mid p_k(\overrightarrow{x}) \in T_P^+ \uparrow \alpha$, $c \mid Q_u\ matchNotP_k(\overrightarrow{x}, \overrightarrow{msg}) \in T_{P_S}^+ \uparrow \alpha$ 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.*, $c \mid Q_u matchP_k(\overrightarrow{x}, \overrightarrow{msg})$, $c \mid Q_u matchNotP_k(\overrightarrow{x}, \overrightarrow{msg})$, $c \mid pesQ_u(\overrightarrow{x})$ 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 $Q_u matchP_k(\overrightarrow{x}, \overrightarrow{msg})\theta$ atom that is added in this step: $Q_u matchP_k(\overrightarrow{x^k_{m_k+j}}, \overrightarrow{msg}) \leftarrow pesQ_1(\overrightarrow{y_1})$, ...,
$pesQ_o(\overrightarrow{y_o})$, $\neg optQ_{o+1}(\overrightarrow{y_{o+1}})$, ..., $\neg optQ_{o+s}(\overrightarrow{y_{o+s}})$,
$Q_u matchP_1(\overrightarrow{x_{1,1}}, \overrightarrow{msg})$, ...,
$Q_u matchP_k(\overrightarrow{x_{m_k,k}}, \overrightarrow{msg})$,
$Q_u matchNotP_k(\overrightarrow{x_{m_k+1,k}}, \overrightarrow{msg})$, ...,
$Q_u matchNotP_k(\overrightarrow{x_{m_k+(j-1),k}}, \overrightarrow{msg})$,
$Q_u matchNotP_k(\overrightarrow{x_{m_k+(j+1),k}}, \overrightarrow{msg})$, ...,
$Q_u matchNotP_t(\overrightarrow{x_{m_t+n_t,t}}, \overrightarrow{msg})$, $\overrightarrow{x_{i,k'}} \neq \overrightarrow{x_{m'_k+j,k'}}$,
$i \in [1, m'_k]$, $j \in [1, n'_k]$, $k' \in$[1,t], c.

The above clause is derived from the following clause in Π: $Q_u(\overrightarrow{x}) \leftarrow Q_1(\overrightarrow{y_1}), \ldots, Q_o(\overrightarrow{y_o}), \neg Q_{o+1}(\overrightarrow{y_{o+1}}), \ldots,$
$\neg Q_{o+s}(\overrightarrow{y_{o+s}})$, $p_1(\overrightarrow{x_{1,1}}), \ldots, p_1(\overrightarrow{x_{m_1,1}})$, $\neg p_1(\overrightarrow{x_{m_1+1,1}})$,
$\ldots, \neg p_1(\overrightarrow{x_{m_1+n_1,1}})$, ..., $p_k(\overrightarrow{x_{1,k}}), \ldots, p_k(\overrightarrow{x_{m_k,k}})$,
$\neg p_k(\overrightarrow{x_{m_k+1,k}}), \ldots, \neg p_k(\overrightarrow{x_{m_k+n_k,k}}), \ldots, p_t(\overrightarrow{x_{1,t}})$,
$\ldots, p_t(\overrightarrow{x_{m_t,t}}), \neg p_t(\overrightarrow{x_{m_t+1,t}}), \ldots, \neg p_t(\overrightarrow{x_{m_t+n_t,t}})$, c.

We wish to show that this clause generates $c \mid Q_u(\overrightarrow{x}) \in T_P^+ \uparrow \beta$ for some β. Since each $pesQ$ constrained atom corresponding to the literal in the body of $Q_u matchP_k(\overrightarrow{x}, \overrightarrow{msg})$ must be in $T_{P_S}^+ \uparrow \alpha$ and each $optQ$ constrained atom in $T_{P_S}^- \uparrow \alpha$, by induction hypothesis each c.atom (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 $c \mid Q_u matchP_k(\overrightarrow{x_{i,k}}, \overrightarrow{msg})$, 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 induction 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 $c \mid Q_u matchNotP_k(\overrightarrow{x_{m_k+j,k}}, \overrightarrow{msg})$ 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 induction hypothesis can be shown for $c \mid Q_u matchNotP_k(\overrightarrow{x}, \overrightarrow{msg})$ 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 induction hypothesis next. As every $pesQ$ literal is defined as $pesQ_u(\overrightarrow{x}) \leftarrow Q_u matchP_k(\overrightarrow{x}, \overrightarrow{msg}), Q_u matchNotP_k(\overrightarrow{x}, \overrightarrow{msg})$ 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 induction hypothesis, *i.e.*, $c \mid optQ_u(\overrightarrow{x}) \in T_{P_S}^-\uparrow\alpha + 1$ entails $c \mid Q_u(\overrightarrow{x}) \in \uparrow\gamma$. Since $c \mid optQ_u(\overrightarrow{x})$ 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 $c \mid optQ_u(\overrightarrow{x})$ literal's addition to $T_{P_S}^+\uparrow\alpha + 1$. By the induction hypothesis, it can be easily shown that either for some $k \in [1, o]$, $c \mid Q_k(\overrightarrow{x_k}) \in \uparrow\gamma$ or for some $k \in [o, s]$, $c \mid Q_k(\overrightarrow{x_k}) \in T_P^+\uparrow\gamma$ for every clause defining $Q_u(\overrightarrow{x})$.

For the limit ordinal case $c \mid matchP_k(\overrightarrow{x}, \overrightarrow{msg}) \in T_{P_S}^+\uparrow\alpha$ (resp. $c \mid matchNotP_k(\overrightarrow{x}, \overrightarrow{msg})$, $c \mid pesQ_u(\overrightarrow{x})$) entails that the c.atom $c \mid matchP_k(\overrightarrow{x}, \overrightarrow{msg}) \in T_{P_S}^+ \uparrow \beta$ (resp. $c \mid matchNotP_k(\overrightarrow{x}, \overrightarrow{msg})$, $c \mid pesQ_u(\overrightarrow{x})$) for some $\beta < \alpha$, by construction. Also, $c \mid optQ_u(\overrightarrow{x}) \in T_{P_S}^- \uparrow \alpha$ entails $c \mid optQ_u(\overrightarrow{x}) \in T_{P_S}^- \uparrow \beta$ for some $\beta < \alpha$, by construction. The induction hypothesis now applies, giving the desired result.

($\Leftarrow$) Here we show that if $c \mid Q_u(\overrightarrow{x}) \in T_P^+\uparrow\omega$, it entails $c \mid pesQ_u(\overrightarrow{x}) \in T_{P_S}^+ \uparrow \omega$. If there is a $c \mid Q_u(\overrightarrow{x}) \in T_P^+ \uparrow \omega$ 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 $c \mid Q_u(\overrightarrow{x}) \in$

$T_P^+\uparrow\omega$. 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 |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 |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 definition 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. Since $pesQ_u(\overrightarrow{x})$ rule in Π_{suf} is constructed by pairing corresponding $matchP$ and $matchNotP$ predicates (*i.e.*, $matchP_k(\overrightarrow{x_k}, \overrightarrow{msg})$ and $matchNotP_k(\overrightarrow{x_k}, \overrightarrow{msg})$), 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 induction 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:

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

This clause is derived from the following general clause in Π: $Q_u(\overrightarrow{x}) \leftarrow Q_1(\overrightarrow{y_1})$,…, $Q_o(\overrightarrow{y_o})$, $\neg Q_{o+1}(\overrightarrow{y_{o+1}})$,…,
$\neg Q_{o+s}(\overrightarrow{y_{o+s}})$, $p_1(\overrightarrow{x_{1,1}})$,…, $p_1(\overrightarrow{x_{m_1,1}})$, $\neg p_1(\overrightarrow{x_{m_1+1,1}})$,
…,$\neg p_1(\overrightarrow{x_{m_1+n_1,1}})$, …, $p_k(\overrightarrow{x_{1,k}})$,…,$p_k(\overrightarrow{x_{m_k,k}})$,
$\neg p_k(\overrightarrow{x_{m_k+1,k}})$,…,$\neg p_k(\overrightarrow{x_{m_k+n_k,k}})$,…, $p_t(\overrightarrow{x_{1,t}})$,
…,$p_t(\overrightarrow{x_{m_t,t}})$,$\neg p_t(\overrightarrow{x_{m_t+1,t}})$,…,$\neg p_t(\overrightarrow{x_{m_t+n_t,t}})$, c.

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 induction 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 |$Q_{o+j}(\overrightarrow{x}) \notin T_P^+\uparrow\gamma$. Hence, for some definition of private predicates, c |$Q_{o+j}(\overrightarrow{x}) \in \uparrow\gamma$. Finally, since $\overrightarrow{y_{i,k'}} \neq \overrightarrow{y_{m_{k'}+j,k'}}$, $i \in [1, m_{k'}]$, $j \in [1, n_{k'}]$, $k' \in [1, t']$, 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 induction 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. Since 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 induction hypothesis. Similarly, since 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 be easily 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. □