Dynamic Enforcement of Knowledge-based Security Policies

Piotr Mardziel, Stephen Magill, Michael Hicks
University of Maryland, College Park

Mudhakar Srivatsa
IBM T.J. Watson Research Laboratory

***Abstract*—This paper explores the idea of *knowledge-based security policies*, which are used to decide whether to answer queries over secret data based on an estimation of the querier's (possibly increased) knowledge given the results. Limiting knowledge is the goal of existing information release policies that employ mechanisms such as noising, anonymization, and redaction. Knowledge-based policies are more general: they increase flexibility by not fixing the means to restrict information flow. We enforce a knowledge-based policy by explicitly tracking a model of a querier's belief about secret data, represented as a probability distribution, and denying any query that could increase knowledge above a given threshold. We implement query analysis and belief tracking via abstract interpretation using a novel *probabilistic polyhedral* domain, whose design permits trading off precision with performance while ensuring estimates of a querier's knowledge are sound. Experiments with our implementation show that several useful queries can be handled efficiently, and performance scales far better than would more standard implementations of probabilistic computation based on sampling.**

***Keywords*-privacy, abstract interpretation, probabilistic polyhedron, knowledge-based security**

I. Introduction

Facebook, Twitter, Flickr, and other successful on-line services enable users to easily foster and maintain relationships by sharing information with friends and fans. These services store users' personal information and use it to customize the user experience and to generate revenue. For example, Facebook third-party applications are granted access to a user's "basic" data (which includes name, profile picture, gender, networks, user ID, and list of friends [1]) to implement services like birthday announcements and horoscopes, while Facebook selects ads based on age, gender, and even sexual preference [2]. Unfortunately, once personal information is collected, users have limited control over how it is used. For example, Facebook's EULA grants Facebook a non-exclusive license to any content a user posts [3]. MySpace, another social network site, has recently begun to sell its users' data [4].

Some researchers have proposed that, to keep tighter control over their data, users could use a storage server (e.g., running on their home network) that handles personal data requests, and only responds when a request is deemed safe [5], [6]. The question is: which requests are safe? While deferring to user-defined access control policies seems an obvious approach, such policies are unnecessarily restrictive when the goal is to maximize the customized personal experience. To see why, consider two example applications: a horoscope or "happy birthday" application that operates on birth month and day, and a music recommendation algorithm that considers birth year (age). Access control at the granularity of the entire birth date could preclude both of these applications, while choosing only to release birth year or birth day precludes access to one application or the other. But in fact the user may not care much about these particular bits of information, but rather about what can be deduced from them. For example, it has been reported that zip code, birth date, and gender are sufficient information to uniquely identify 63% of Americans in the 2000 U.S. census [7]. So the user may be perfectly happy to reveal any one of these bits of information in its entirety as long as a querier gains no better than a $1/n$ chance to guess the entire group, for some parameter n.

This paper explores the design and implementation for enforcing what we call *knowledge-based security policies*. In our model, a user U's agent responds to queries involving secret data. For each querying principal Q, the agent maintains a probability distribution over U's secret data, representing Q's *belief* of the data's likely values. For example, to mediate queries from a social networking site X, user U's agent may model X's otherwise uninformed knowledge of U's birthday according to a likely demographic: the birth month and day are uniformly distributed, while the birth year is most likely between 1956 and 1992 [8]. Each querier Q is also assigned a knowledge-based policy, expressed as a set of thresholds, each applying to a different group of (potentially overlapping) data. For example, U's policy for X might be a threshold of $1/100$ for the entire tuple $(birthdate, zipcode, gender)$, and $1/5$ for just birth date. U's agent refuses any queries that it determines could increase Q's ability to guess a secret above the assigned threshold. If deemed safe, U's agent returns the query's (exact) result and updates Q's modeled belief appropriately. (We touch upon the risk of colluding queriers shortly.)

To implement our model, we need (1) an algorithm to check whether answering a query could violate a knowledge-based policy, (2) a method for revising a querier's belief according to the answer that is given, and (3) means to implement (1) and (2) efficiently. We build on the work of Clarkson et al. [9] (reviewed in Section III), which works out the theoretical basis for (2). The main contributions of this paper, therefore, in addition to the idea of knowledge-based policies, are our solutions to problems (1) and (3).

Given a means to revise querier beliefs based on prior answers, it seems obvious how to check that a query does not reveal too much: U runs the query, tentatively revises Q's belief based on the result, and then responds with the answer only if Q's revised belief about the secrets does not exceed the prescribed thresholds. Unfortunately, with this approach the decision to deny depends on the actual secret, so a rejection could leak information. We give an example in the next section that shows how the entire secret could be revealed. Therefore, we propose that a query should be rejected if there exists *any* possible secret value that could induce an output whereby the revised belief would exceed the threshold. This idea is described in detail in Section IV.

To implement belief tracking and revision, our first thought was to use languages for probabilistic computation and conditioning, which provide the foundational elements of the approach. Languages we know of—IBAL [10], Probabilistic Scheme [11], and several other systems [12], [13], [14]—are implemented using sampling. Unfortunately, we found these implementations to be inadequate because they either underestimate the querier's knowledge when sampling too little, or run too slowly when the state space is large.

Instead of using sampling, we have developed an implementation based on abstract interpretation. In Section V we develop a novel abstract domain of *probabilistic polyhedra*, which extends the standard convex polyhedron abstract domain [15] with measures of probability. We represent beliefs as a set of probabilistic polyhedra (as developed in Section VI). While some prior work has explored probabilistic abstract interpretation [16], this work does not support belief revision, which is required to track how observation of outputs affects a querier's belief. Support for revision requires that we maintain both under- and over-approximations of the querier's belief, whereas [16] deals only with over-approximation. We have developed an implementation of our approach based on Parma [17] and LattE [18], which we present in Section VII along with some experimental measurements of its performance. We find that while the performance of Probabilistic Scheme degrades significantly as the input space grows, our implementation scales much better, and can be orders of magnitude faster.

Knowledge-based policies aim to ensure that an attacker's knowledge of a secret does not increase much when learning the result of a query. Much prior work aims to enforce similar properties by tracking information leakage quantitatively [19], [20], [21], [22], [23]. Our approach is more precise (but also more resource-intensive) because it maintains an on-line model of adversary knowledge. An alternative to knowledge-based privacy is *differential privacy* [24] (DP), which requires that a query over a database of individuals' records produces roughly the same answer whether a particular individual's data is in the database or not—the possible knowledge of the querier, and the impact of the query's result on it, need not be directly considered. As such, DP avoids the danger of mismodeling a querier's knowledge and as a result inappropriately releasing information. DP also ensures a high degree of compositionality, which provides some assurance against collusion. However, DP applies once an individual has released his personal data to a trusted third party's database, a release we are motivated to avoid. Moreover, applying DP to queries over an individual's data, rather than a population, introduces so much noise that the results are often useless. We discuss these issues along with other related work in Section VIII.

The next section presents a technical overview of the rest of the paper, whose main results are contained in Sections III–VII, with further discussion and ideas for future work in Sections VIII and IX.

II. Overview

Knowledge-based policies and beliefs. User Bob would like to enforce a knowledge-based policy on his data so that advertisers do not learn too much about him. Suppose Bob considers his birthday of September 27, 1980 to be relatively private; variable $bday$ stores the calendar day (a number between 0 and 364, which for Bob would be 270) and $byear$ stores the birth year (which would be 1980). To $bday$ he assigns a *knowledge threshold* $t_d = 0.2$ stating that he does not want an advertiser to have better than a 20% likelihood of guessing his birth day. To the pair $(bday, byear)$ he assigns a threshold $t_{dy} = 0.05$, meaning he does not want an advertiser to be able to guess the combination of birth day *and* year together with better than a 5% likelihood.

Bob runs an agent program to answer queries about his data on his behalf. This agent models an estimated *belief* of queriers as a probability distribution δ, which is conceptually a map from secret states to positive real numbers representing probabilities (in range $[0, 1]$). Bob's secret state is the pair $(bday = 270, byear = 1980)$. The agent represents a distribution as a set of probabilistic polyhedra. For now, we can think of a probabilistic polyhedron as a standard convex polyhedron C with a probability mass m, where the probability of each integer point contained in C is $m/\#(C)$, where $\#(C)$ is the number of integer points contained in the polyhedron C. Shortly we present a more involved representation.

Initially, the agent might model an advertiser X's belief using the following rectangular polyhedron C, where each point contained in it is considered equally likely ($m = 1$):

$$C = 0 \leq bday < 365, \ 1956 \leq byear < 1993$$

Enforcing knowledge-based policies safely. Suppose X wants to identify users whose birthday falls within the next week, to promote a special offer. X sends Bob's agent the following program.

Example 1.

```
$today$ := 260;
if $bday \geq today \wedge bday < (today + 7)$ then
  $output$ := True;
```

This program refers to Bob's secret variable $bday$, and also uses non-secret variables $today$, which represents the current day and is here set to be 260, and $output$, which is set to True if the user's birthday is within the next seven days (we assume $output$ is initially False).

The agent must decide whether returning the result of running this program will potentially increase X's knowledge about Bob's data above the prescribed threshold. We explain how it makes this determination shortly, but for the present we can see that answering the query is safe: the returned $output$ variable will be False which essentially teaches the querier that Bob's birthday is not within the next week, which still leaves many possibilities. As such, the agent *revises* his model of the querier's belief to be the following *pair* of rectangular polyhedra C_1, C_2, where again all points in each are equally likely ($m_1 \approx 0.726, m_2 \approx 0.274$):

$$C_1 = 0 \leq bday < 260,\ 1956 \leq byear < 1993$$
$$C_2 = 267 \leq bday < 365,\ 1956 \leq byear < 1993$$

Ignoring $byear$, there are 358 possible values for $bday$ and each is equally likely. Thus the probability of any one is $1/358 \approx 0.0028 \leq t_d = 0.2$.

Suppose the next day the same advertiser sends the same program to Bob's user agent, but with $today$ set to 261. Should the agent run the program? At first glance, doing so seems OK. The program will return False, and the revised belief will be the same as above but with constraint $bday \geq 267$ changed to $bday \geq 268$, meaning there is still only a $1/357 = 0.0028$ chance to guess $bday$.

But suppose Bob's birth day was actually 267, rather than 270. The first query would have produced the same revised belief as before, but since the second query would return True (since $bday = 267 < (261{+}7)$), the querier can deduce Bob's birth day exactly: $bday \geq 267$ (from the first query) and $bday < 268$ (from the second query) together imply that $bday = 267$! But the user agent is now stuck: it cannot simply refuse to answer the query, because the querier knows that with $t_d = 0.2$ (or indeed, any reasonable threshold) the only good reason to refuse is when $bday = 267$. As such, refusal essentially tells the querier the answer.

The lesson is that the decision to refuse a query must not be based on the effect of running the query on the actual secret, because then a refusal could leak information. In Section IV we propose that an agent should reject a program if there exists *any* possible secret that could cause a program answer to increase querier knowledge above the threshold. As such we would reject the second query regardless of whether $bday = 270$ or $bday = 267$.

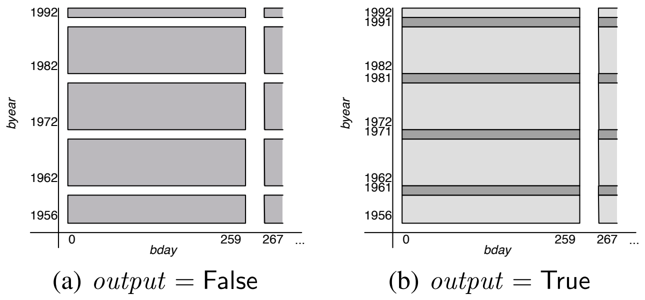


Figure 1. Example 2: most precise revised beliefs

Full probabilistic polyhedra. Now suppose, having run the first query and rejected the second, the user agent receives the following program from X.

Example 2.

```
$age$ := $2011 - byear$;
if $age = 20 \vee ... \vee age = 60$ then
  $output$ := True;
pif 0.1 then $output$ := True;
```

This program attempts to discover whether this year is a "special" year for the given user, who thus deserves a special offer. The program returns True if either the user's age is (or will be) an exact decade, or if the user wins the luck of the draw (one chance in ten), as implemented by the probabilistic if statement.

Running this program reveals nothing about $bday$, but does reveal something about $byear$. In particular, if $output$ = False then the querier knows that $byear \notin \{1991, 1981, 1971, 1961\}$, but all other years are equally likely. We could represent this new knowledge, combined with the knowledge gained from the first query, as shown in Figure 1(a), where each shaded box is a polyhedron containing equally likely points. On the other hand, if $output$ = True then either $byear \in \{1991, 1981, 1971, 1961\}$ or the user got lucky. We represent the querier's knowledge in this case as in Figure 1(b). Darker shading indicates higher probability; thus, all years are still possible, though some are much more likely than others. With the given threshold of $t_{dy} = 0.05$, the agent will permit the query; when $output$ = False, the likelihood of any point in the shaded region is $1/11814$; when $output$ = True, the points in the dark bands are the most likely, with probability $5/13067$. Since both outcomes are possible with Bob's $byear = 1980$, the revised belief will depend on the result of the probabilistic if statement.

This example illustrates a potential problem with the simple representation of probabilistic polyhedra mentioned earlier: when $output$ = False we will jump from using two probabilistic polyhedra to ten, and when $output$ = True we jump to using eighteen. Allowing the number of polyhedra to grow without bound will result in performance problems.

To address this concern, we need a way to abstract our belief representation to be more concise. Section V shows how to represent a probabilistic polyhedron P as a seven-tuple, $(C, \mathrm{s}^{\min}, \mathrm{s}^{\max}, \mathrm{p}^{\min}, \mathrm{p}^{\max}, \mathrm{m}^{\min}, \mathrm{m}^{\max})$ where $\mathrm{s}^{\min}$ and $\mathrm{s}^{\max}$ are lower and upper bounds on the number of points with non-zero probability in the polyhedron C (called the *support points* of C); the quantities $\mathrm{p}^{\min}$ and $\mathrm{p}^{\max}$ are lower and upper bounds on the probability mass per support point; and $\mathrm{m}^{\min}$ and $\mathrm{m}^{\max}$ give bounds on the total probability mass. Thus, polyhedra modeled using the simpler representation (C, m) given earlier are equivalent to ones in the more involved representation with $\mathrm{m}^{\max} = \mathrm{m}^{\min} = m$, $\mathrm{p}^{\max} = \mathrm{p}^{\min} = m/\#(C)$, and $\mathrm{s}^{\max} = \mathrm{s}^{\min} = \#(C)$.

With this representation, we could choose to collapse the sets of polyhedron given in Figure 1. For example, we could represent Figure 1(a) with two probabilistic polyhedra P_1 and P_2 containing polyhedra C_1 and C_2 defined above, respectively, essentially drawing a box around the two groupings of smaller boxes in the figure. The other parameters for P_1 would be as follows:

$$\begin{array}{l} \mathrm{p}_1^{\min} = \mathrm{p}_1^{\max} = 9/135050 \\ \mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = 8580 \\ \mathrm{m}_1^{\min} = \mathrm{m}_1^{\max} = 7722/13505 \end{array}$$

Notice that $\mathrm{s}_1^{\min} = \mathrm{s}_2^{\max} = 8580 < \#(C_1) = 9620$, illustrating that the "bounding box" of the polyhedron covers more area than is strictly necessary. In this representation the probabilities may not be normalized, which improves both performance and precision. For this example, P_2 happens to have $\mathrm{m}_2^{\min} = \mathrm{m}_2^{\max} = 14553/67525$ so we can see $\mathrm{m}_1^{\max} + \mathrm{m}_2^{\max} = (53163/67525) \neq 1$.

If we consider the representation of Figure 1(b) in a similar manner, using the same two polyhedra C_1 and C_2, the other parameters for C_1 are as follows:

$$\begin{array}{ll} \mathrm{p}_1^{\min} = 1/135050 & \mathrm{p}_1^{\max} = 10/135050 \\ \mathrm{s}_1^{\min} = 9620 & \mathrm{s}_1^{\max} = 9620 \\ \mathrm{m}_1^{\min} = 26/185 & \mathrm{m}_1^{\max} = 26/185 \end{array}$$

In this case $\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = \#(C_1)$, meaning that all covered points are possible, but $\mathrm{p}_1^{\min} \neq \mathrm{p}_1^{\max}$ as some points are more probable than others (i.e., those in the darker band).

The key property of probabilistic polyhedra, and a main technical contribution of this paper, is that this abstraction can be used to make sound security policy decisions. To accept a query, we must check that, for all possible outputs, the querier's revised, normalized belief of any of the possible secrets is below the threshold t. In checking whether the revised beliefs in our example are acceptable, the agent will try to find the maximum probability the querier could ascribe to a state, for each possible output. In the case $output =$ True, the most probable points are those in the dark bands, which each have probability mass $10/135050 = \mathrm{p}_1^{\max}$ (the dark bands in P_2 have the same probability). To find the maximum normalized probability of these points, we divide by the minimum possible total mass, as given by the lower bounds in our abstraction. In our example, this results in $\mathrm{p}_1^{\max}/(\mathrm{m}_1^{\min}+\mathrm{m}_2^{\min}) = (10/135050)/(26/185+49/925) \approx 0.0004 \leq t_d = 0.05$.

As just shown, the bound on minimum total mass is needed in order to soundly normalize distributions in our abstraction. The maintenance of such lower bounds on probability mass is a key component of our abstraction that is missing from prior work. Each of the components of a probabilistic polyhedron play a role in producing the lower bound on total mass. While $\mathrm{s}_1^{\min}, \mathrm{s}_1^{\max}, \mathrm{p}_1^{\min}$, and $\mathrm{m}_1^{\max}$ do not play a role in making the final policy decision, their existence allows us to more accurately update belief during the query evaluation that precedes the final policy check. The choice of the number of probabilistic polyhedra to use impacts both precision and performance, so choosing the right number is a challenge. For the examples given in this section, our implementation can often answer queries in a few seconds; details are in Sections V–VII.

Variables	x	$\in$	**Var**
Integers	n	$\in$	$\mathbb{Z}$
Rationals	q	$\in$	$\mathbb{Q}$
Arith.ops	*aop*	::=	$+ \mid \times \mid -$
Rel.ops	*relop*	::=	$\leq \mid < \mid = \mid \neq \mid \cdots$
Arith.exps	E	::=	$x \mid n \mid E_1 \; aop \; E_2$
Bool.exps	B	::=	$E_1 \; relop \; E_2 \mid$
			$B_1 \wedge B_2 \mid B_1 \vee B_2 \mid \neg B$
Statements	S	::=	skip $\mid x$:= $E \mid$
			if B then S_1 else $S_2 \mid$
			pif q then S_1 else $S_2 \mid$
			S_1 ; $S_2 \mid$ while B do S

Figure 2. Core language syntax

III. Tracking beliefs

This section reviews Clarkson et al.'s method of revising a querier's belief of the possible valuations of secret variables based on the result of a query involving those variables [9].

A. Core language

The programming language we use for queries is given in Figure 2. A computation is defined by a statement S whose standard semantics can be viewed as a relation between states: given an input state σ, running the program will produce an output state σ'. States are maps from variables to integers:

$$\sigma, \tau \in \mathbf{State} \overset{\text{def}}{=} \mathbf{Var} \rightarrow \mathbb{Z}$$

Sometimes we consider states with domains restricted to a subset of variables V, in which case we write $\sigma_V \in \mathbf{State}_V \overset{\text{def}}{=} V \rightarrow \mathbb{Z}$. We may also *project* states to a set of variables V:

$$\sigma \upharpoonright V \overset{\text{def}}{=} \lambda x \in \mathbf{Var}_V.\, \sigma(x)$$

$$\begin{array}{rcl}
[\![\mathsf{skip}]\!]\delta & = & \delta \\
[\![x := E]\!]\delta & = & \delta\,[x \to E] \\
[\![\mathsf{if}\ B\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2]\!]\delta & = & [\![S_1]\!](\delta|B) + [\![S_2]\!](\delta|\neg B) \\
[\![\mathsf{pif}\ q\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2]\!]\delta & = & [\![S_1]\!](q \cdot \delta) + [\![S_2]\!]((1-q)\cdot\delta) \\
[\![S_1\ ;\ S_2]\!]\delta & = & [\![S_2]\!]([\![S_1]\!]\delta) \\
[\![\mathsf{while}\ B\ \mathsf{do}\ S]\!] & = & \mathrm{lfp}\,[\lambda f:\ \mathbf{Dist} \to \mathbf{Dist}.\ \lambda\delta. \\
 & & \quad f\,([\![S]\!](\delta|B)) + (\delta|\neg B)]
\end{array}$$

where

$$\begin{array}{ll}
\delta\,[x \to E] & \stackrel{\text{def}}{=} \lambda\sigma.\ \sum_{\tau\ |\ \tau[x \to [\![E]\!]\tau] = \sigma} \delta(\tau) \\
\delta_1 + \delta_2 & \stackrel{\text{def}}{=} \lambda\sigma.\ \delta_1(\sigma) + \delta_2(\sigma) \\
\delta|B & \stackrel{\text{def}}{=} \lambda\sigma.\ \mathbf{if}\ [\![B]\!]\sigma\ \mathbf{then}\ \delta(\sigma)\ \mathbf{else}\ 0 \\
p \cdot \delta & \stackrel{\text{def}}{=} \lambda\sigma.\ p \cdot \delta(\sigma)
\end{array}$$

Figure 3. Probabilistic semantics for the core language

The language is essentially standard. We limit the form of expressions to support our abstract interpretation-based semantics (Section V). The semantics of the statement form pif q then S_1 else S_2 is non-deterministic: the result is that of S_1 with probability q, and S_2 with probability $1 - q$.

B. Probabilistic semantics for tracking beliefs

To enforce a knowledge-based policy, a user agent must be able to estimate what a querier could learn from the output of his query. To do this, the agent keeps a distribution δ that represents the querier's *belief* of the likely valuations of the user's secrets. More precisely, a distribution is a map from states to positive real numbers, interpreted as probabilities (in range $[0, 1]$).

$$\delta \in \mathbf{Dist} \stackrel{\text{def}}{=} \mathbf{State} \to \mathbb{R}+$$

We sometimes focus our attention on distributions over states of a fixed set of variables V, in which case we write $\delta_V \in \mathbf{Dist}_V$ to mean $\mathbf{State}_V \to \mathbb{R}+$. Projecting distributions onto a set of variables is as follows:[1]

$$\delta \restriction V \stackrel{\text{def}}{=} \lambda\sigma_V \in \mathbf{State}_V.\ \sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta(\sigma')$$

The *mass* of a distribution, written $\|\delta\|$ is the sum of the probabilities ascribed to states, $\sum_\sigma \delta(\sigma)$. A *normalized distribution* is one such that $\|\delta\| = 1$. A normalized distribution can be constructed by scaling a distribution according to its mass:

$$\mathrm{normal}(\delta) \stackrel{\text{def}}{=} \frac{1}{\|\delta\|} \cdot \delta$$

The *support* of a distribution is the set of states which have non-zero probability: *support*$(\delta) \stackrel{\text{def}}{=} \{\sigma \mid \delta(\sigma) > 0\}$.

The agent evaluates a query in light of the querier's initial belief using a probabilistic semantics. Figure 3 defines a semantic function $[\![\cdot]\!]$ whereby $[\![S]\!]\delta = \delta'$ indicates that, given an input distribution δ, the semantics of program S is the output distribution δ'. The semantics is defined in terms of operations on distributions, including *assignment* $\delta\,[v \to E]$ (used in the rule for $v := E$), *conditioning* $\delta|B$ and *addition* $\delta_1 + \delta_2$ (used in the rule for if), and *scaling* $q \cdot \delta$ where q is a rational (used for pif). The semantics is standard (cf. Clarkson et al. [9]).

[1]The notation $\sum_{x|\pi} \rho$ can be read *ρ is the sum over all x such that formula π is satisfied* (where x is bound in ρ and π).

C. Belief and security

Clarkson et al. [9] describe how a belief about possible values of a secret, expressed as a probability distribution, can be revised according to an experiment using the actual secret. Such an experiment works as follows.

The values of the set of secret variables H are given by the hidden state σ_H. The attacker's initial belief as to the possible values of σ_H is represented as a distribution δ_H. A query is a program S that makes use of variables H and possibly other, non-secret variables from a set L; the final values of L, after running S, are made visible to the attacker. Let σ_L be an arbitrary initial state of these variables such that *domain*$(\sigma_L) = L$. Then we take the following steps:

Step 1. Evaluate S probabilistically using the attacker's belief about the secret to produce an output distribution δ', which amounts to the attacker's prediction of the possible output states. This is computed as $\delta' = [\![S]\!]\delta$, where δ, a distribution over variables $H \uplus L$, is defined as $\delta = \delta_H \times \dot{\sigma}_L$. Here, we make use of the distribution product operator and point operator. That is, given δ_1, δ_2, which are distributions over states having disjoint domains, the *distribution product* is

$$\delta_1 \times \delta_2 \stackrel{\text{def}}{=} \lambda(\sigma_1, \sigma_2).\ \delta_1(\sigma_1) \cdot \delta_2(\sigma_2)$$

where (σ_1, σ_2) is the "concatenation" of the two states, which is itself a state and is well-defined because the two states' domains are disjoint. And, given a state σ, the *point distribution* $\dot{\sigma}$ is a distribution in which only σ is possible:

$$\dot{\sigma} \stackrel{\text{def}}{=} \lambda\tau.\ \mathbf{if}\ \sigma = \tau\ \mathbf{then}\ 1\ \mathbf{else}\ 0$$

Thus, the initial distribution δ is the attacker's belief about the secret variables combined with an arbitrary valuation of the public variables.

Step 2. Using the actual secret σ_H, evaluate S "concretely" to produce an output state $\hat{\sigma}_L$, in three steps. First, we have $\hat{\delta}' = [\![S]\!]\hat{\delta}$, where $\hat{\delta} = \dot{\sigma}_H \times \dot{\sigma}_L$. Second, we have $\hat{\sigma} \in \Gamma(\hat{\delta})$ where Γ is a sampling operator that produces a state σ from the domain of a distribution δ with probability $\delta(\sigma)/\|\delta\|$. Finally, we extract the attacker-visible output of the sampled state by projecting away the high variables: $\hat{\sigma}_L = \hat{\sigma} \restriction L$.

Step 3. Revise the attacker's initial belief δ_H according to the observed output $\hat{\sigma}_L$, yielding a new belief $\hat{\delta}_H = \delta'|\hat{\sigma}_L \restriction H$. Here, δ' is *conditioned* on the output $\hat{\sigma}_L$, which yields a new distribution, and this distribution is then projected to the variables H. The conditioning operation is defined as follows:

$$\delta|\sigma_V \overset{\text{def}}{=} \lambda\sigma.\ \textbf{if } \sigma \upharpoonright V = \sigma_V \textbf{ then } \delta(\sigma) \textbf{ else } 0$$

Note that this protocol assumes that S always terminates and does not modify the secret state. The latter assumption can be eliminated by essentially making a copy of the state before running the program, while eliminating the former depends on the observer's ability to detect nontermination [9].

IV. Enforcing knowledge-based policies

When presented with a query over a user's data σ_H, the user's agent should only answer the query if doing so will not reveal too much information. More precisely, given a query S, the agent will return the public output σ_L resulting from running S on σ_H if the agent deems that from this output the querier cannot guess the secret state σ_H beyond some level of doubt, identified by a threshold t. If this threshold could be exceeded, then the agent declines to run S. We call this security check *knowledge threshold security*.

Definition 3 (Knowledge Threshold Security)**.** Let $\delta' = [\![S]\!]\delta$, where δ is the model of the querier's initial belief. Then query S is *threshold secure* iff for all $\sigma_L \in$ *support*$(\delta' \upharpoonright L)$ and all $\sigma'_H \in \textbf{State}_H$ we have $(\text{normal}((\delta'|\sigma_L) \upharpoonright H))(\sigma'_H) \leq t$ for some threshold t.

This definition can be related to the experiment protocol defined in Section III-C. First, δ' in the definition is the same as δ' computed in the first step of the protocol. Step 2 in the protocol produces a concrete output $\hat{\sigma}_L$ based on executing S on the actual secret σ_H, and Step 3 revises the querier's belief based on this output. Definition 3 generalizes these two steps: instead of considering a single concrete output based on the actual secret it considers *all possible* concrete outputs, as given by *support*$(\delta' \upharpoonright L)$, and ensures that the revised belief in each case for *all possible* secret states must assign probability no greater than t.

This definition considers a threshold for the whole secret state σ_H. As described in Section II we can also enforce thresholds over portions of a secret state. In particular, a threshold that applies only to variables $V \subseteq H$ requires that all $\sigma'_V \in \textbf{State}_V$ result in $(\text{normal}(\delta'|\sigma_L \upharpoonright V))(\sigma'_V) \leq t$.

The two "foralls" in the definition are critical for ensuring security. The reason was shown by the first example in Section II: If we used the flawed approach of just running the experiment protocol and checking if $\hat{\delta}_H(\sigma_H) > t$ then rejection depends on the value of the secret state and could reveal information about it. The more general policy $\forall\sigma_L \in$ *support*$(\delta' \upharpoonright L).\,(\text{normal}(\delta'|\sigma_L \upharpoonright H))(\sigma_H) \leq t$, would sidestep the problem in the example, but this policy could still reveal information because it, too, depends on the actual secret σ_H. Definition 3 avoids any inadvertent information leakage because rejection is not based on the actual secret: if there exists *any* secret such that a possible output would reveal too much, the query is rejected. Definition 3 resembles, but is stronger than, *min-entropy*, as the security decision is based on the most likely secret from the attacker's point of view [20]; further details are given in Section VIII.

V. Belief revision via abstract interpretation

Consider how we might implement belief tracking and revision to enforce the threshold security property given in Definition 3. A natural choice would be to evaluate queries using a probabilistic programming language with support for conditioning; examples are IBAL [10], Probabilistic Scheme [11], and several others [12], [13], [14]. In these languages, probabilistic evaluation is achieved by enumerating inputs (sampling). Probabilities are associated with each input and tracked during execution. As more inputs are enumerated, a more complete view of the output distribution emerges. Unfortunately, to get an accurate estimate of the revised distribution following an output observation, one must enumerate the entire input space, which could be quite large. If insufficient coverage is achieved, then the threshold check in Definition 3 could either be unsound or excessively conservative, depending in which direction an implementation errs.

To avoid sampling, we have developed a new means to perform probabilistic computation based on abstract interpretation. In this approach, execution time depends on the complexity of the query rather than the size of the input space. In the next two sections, we present two abstract domains. This section presents the first, denoted $\mathbb{P}$, where an abstract element is a single *probabilistic polyhedron*, which is a convex polyhedron [15] with information about the probabilities of its points. Because using a single polyhedron will accumulate imprecision after multiple queries, in our implementation we actually use a different domain, denoted $\mathcal{P}_n(\mathbb{P})$, for which an abstract element consists of a set of at most n probabilistic polyhedra (whose construction is inspired by powersets of polyhedra [25], [26]). This domain, described in the next section, allows us to retain precision at the cost of increased execution time. By adjusting n, the user can trade off efficiency and precision.

A. Polyhedra

We first review *convex polyhedra*, a common technique for representing sets of program states. We use the metavariables β, β_1, β_2, etc. to denote linear inequalities. We write $fv(\beta)$ to be the set of variables occurring in β; we also extend this to sets, writing $fv(\{\beta_1, \ldots, \beta_n\})$ for $fv(\beta_1) \cup \ldots \cup fv(\beta_n)$.

Definition 4. A *convex polyhedron* $C = (B, V)$ is a set of linear inequalities $B = \{\beta_1, \ldots, \beta_m\}$, interpreted conjunctively, over dimensions V. We write $\mathbb{C}$ for the set of all convex polyhedra. A polyhedron C represents a set of states, denoted $\gamma_{\mathbb{C}}(C)$, as follows, where $\sigma \models \beta$ indicates that the state σ satisfies the inequality β.

$$\gamma_{\mathbb{C}}((B,V)) \stackrel{\text{def}}{=} \{\sigma \mid \mathit{domain}(\sigma) = V \ , \ \forall \beta \in B.\ \sigma \models \beta\}$$

Naturally we require that $\mathit{fv}(\{\beta_1, \ldots, \beta_n\}) \subseteq V$. We write $\mathit{fv}((B,V))$ to denote the set of variables V of a polyhedron.

Given a state σ and an ordering on the variables in $\mathit{domain}(\sigma)$, we can view σ as a point in an N-dimensional space, where $N = |\mathit{domain}(\sigma)|$. The set $\gamma_{\mathbb{C}}(C)$ can then be viewed as the integer-valued lattice points in an N-dimensional polyhedron. Due to this correspondence, we use the words *point* and *state* interchangeably. We will sometimes write linear equalities $x = f(\vec{y})$ as an abbreviation for the pair of inequalities $x \leq f(\vec{y})$ and $x \geq f(\vec{y})$.

Let $C = (B,V)$. Convex polyhedra support the following operations.

• Polyhedron size, or $\#(C)$, is the number of integer points in the polyhedron, i.e., $|\gamma_{\mathbb{C}}(C)|$. We will always consider bounded polyhedra when determining their size, ensuring that $\#(C)$ is finite.

• Expression evaluation, $\langle\langle B \rangle\rangle\, C$ returns a convex polyhedron containing at least the points in C that satisfy B.

• Expression count, $C \# B$ returns an upper bound on the number of integer points in C that satisfy B. (It may be more precise than $\#(\langle\langle B \rangle\rangle\, C)$.)

• Meet, $C_1 \sqcap_{\mathbb{C}} C_2$ is the convex polyhedron containing exactly the set of points in the intersection of $\gamma_{\mathbb{C}}(C_1), \gamma_{\mathbb{C}}(C_2)$.

• Join, $C_1 \sqcup_{\mathbb{C}} C_2$ is the smallest convex polyhedron containing both $\gamma(C_1)$ and $\gamma(C_2)$.

• Comparison, $C_1 \sqsubseteq_{\mathbb{C}} C_2$ is a partial order whereby $C_1 \sqsubseteq_{\mathbb{C}} C_2$ if and only if $\gamma(C_1) \subseteq \gamma(C_2)$.

• Affine transform, $C\,[x \to E]$, where $x \in \mathit{fv}(C)$, computes an affine transformation of C. This scales the dimension corresponding to x by the coefficient of x in E and shifts the polyhedron. For example, $(\{x \leq y, y = 2z\}, V)\,[y \to z + y]$ evaluates to $(\{x \leq y - z, y - z = 2z\}, V)$.

• Forget, $\mathrm{f}_x(C)$, projects away x. That is, $\mathrm{f}_x(C) = \pi_{\mathit{fv}(C)-\{x\}}(C)$, where $\pi_V(C)$ is a polyhedron C' such that $\gamma_{\mathbb{C}}(C') = \{\sigma \mid \sigma' \in \gamma_{\mathbb{C}}(C) \wedge \sigma = \sigma' \upharpoonright V\}$. So $C' = \mathrm{f}_x(C)$ implies $x \notin fv(C')$.

We write $\mathit{isempty}(C)$ *iff* $\gamma_{\mathbb{C}}(C) = \emptyset$.

B. Probabilistic Polyhedra

We take this standard representation of sets of program states and extend it to a representation for sets of distributions over program states. We define *probabilistic polyhedra*, the core element of our abstract domain, as follows.

Definition 5. A *probabilistic polyhedron* P is a tuple $(C, \mathrm{s}^{\min}, \mathrm{s}^{\max}, \mathrm{p}^{\min}, \mathrm{p}^{\max}, \mathrm{m}^{\min}, \mathrm{m}^{\max})$. We write $\mathbb{P}$ for the set of probabilistic polyhedra. The quantities $\mathrm{s}^{\min}$ and $\mathrm{s}^{\max}$ are lower and upper bounds on the number of support points in the polyhedron C. The quantities $\mathrm{p}^{\min}$ and $\mathrm{p}^{\max}$ are lower and upper bounds on the probability mass *per support point*. The $\mathrm{m}^{\min}$ and $\mathrm{m}^{\max}$ components give bounds on the total probability mass. Thus P represents the *set* of distributions $\gamma_{\mathbb{P}}(P)$ defined below.

$$\begin{aligned}\gamma_{\mathbb{P}}(P) \stackrel{\text{def}}{=} \{\delta \mid\ & \mathit{support}(\delta) \subseteq \gamma_{\mathbb{C}}(C) \wedge \\ & \mathrm{s}^{\min} \leq |\mathit{support}(\delta)| \leq \mathrm{s}^{\max} \wedge \\ & \mathrm{m}^{\min} \leq \|\delta\| \leq \mathrm{m}^{\max} \wedge \\ & \forall \sigma \in \mathit{support}(\delta).\ \mathrm{p}^{\min} \leq \delta(\sigma) \leq \mathrm{p}^{\max}\}\end{aligned}$$

We will write $\mathit{fv}(P) \stackrel{\text{def}}{=} \mathit{fv}(C)$ to denote the set of variables used in the probabilistic polyhedron.

Note the set $\gamma_{\mathbb{P}}(P)$ is singleton exactly when $\mathrm{s}^{\min} = \mathrm{s}^{\max} = \#(C)$ and $\mathrm{p}^{\min} = \mathrm{p}^{\max}$, and $\mathrm{m}^{\min} = \mathrm{m}^{\max}$. In such a case $\gamma_{\mathbb{P}}(P)$ is the uniform distribution where each state in $\gamma_{\mathbb{C}}(C)$ has probability $\mathrm{p}^{\min}$. Distributions represented by a probabilistic polyhedron are not necessarily normalized (as was true in Section III-B). In general, there is a relationship between $\mathrm{p}^{\min}, \mathrm{s}^{\min}$, and $\mathrm{m}^{\min}$, in that $\mathrm{m}^{\min} \geq \mathrm{p}^{\min} \cdot \mathrm{s}^{\min}$ (and $\mathrm{m}^{\max} \leq \mathrm{p}^{\max} \cdot \mathrm{s}^{\max}$), and the combination of the three can yield more information than any two in isolation.

Our convention will be to use C_1, $\mathrm{s}_1^{\min}$, $\mathrm{s}_1^{\max}$, etc. for the components associated with probabilistic polyhedron P_1 and to use subscripts to name different probabilistic polyhedra.

Distributions are ordered point-wise [9]. That is, $\delta_1 \leq \delta_2$ if and only if $\forall \sigma.\ \delta_1(\sigma) \leq \delta_2(\sigma)$. For our abstract domain, we say that $P_1 \sqsubseteq_{\mathbb{P}} P_2$ if and only if $\forall \delta_1 \in \gamma_{\mathbb{P}}(P_1).\ \exists \delta_2 \in \gamma_{\mathbb{P}}(P_2).\ \delta_1 \leq \delta_2$. Testing $P_1 \sqsubseteq_{\mathbb{P}} P_2$ mechanically is non-trivial, but is unnecessary in our semantics. Rather, we need to test whether a distribution represents only the zero distribution $0_{\mathbf{Dist}} \stackrel{\text{def}}{=} \lambda\sigma.0$ in order to see that a fixed point for evaluating $\langle\langle \mathsf{while}\ B\ \mathsf{do}\ S \rangle\rangle\, P$ has been reached. Intuitively, no further iterations of the loop need to be considered once the probability mass flowing into the n^{th} iteration is zero. This condition can be detected as follows:

$$\begin{aligned}&\mathit{iszero}(P) \stackrel{\text{def}}{=} \\ &\quad \mathrm{s}^{\min} = \mathrm{s}^{\max} = 0 \wedge \mathrm{m}^{\min} = 0 \leq \mathrm{m}^{\max} \\ &\vee\ \mathrm{m}^{\min} = \mathrm{m}^{\max} = 0 \wedge \mathrm{s}^{\min} = 0 \leq \mathrm{s}^{\max} \\ &\vee\ \mathit{isempty}(C) \wedge \mathrm{s}^{\min} = 0 \leq \mathrm{s}^{\max} \wedge \mathrm{m}^{\min} = 0 \leq \mathrm{m}^{\max} \\ &\vee\ \mathrm{p}^{\min} = \mathrm{p}^{\max} = 0 \wedge \mathrm{s}^{\min} = 0 \leq \mathrm{s}^{\max} \wedge \mathrm{m}^{\min} = 0 \leq \mathrm{m}^{\max}\end{aligned}$$

If $\mathit{iszero}(P)$ holds, it is the case that $\gamma_{\mathbb{P}}(P) = \{0_{\mathbf{Dist}}\}$. Note that having a more conservative definition of this function (which holds for fewer probabilistic polyhedra) would be reasonable since it would simply mean our analysis would terminate less often than it could, with no effect on security. More details are given in our technical report [27].

In a standard abstract domain, termination of the fixed point computation for loops is often ensured by use of a widening operator. This allows abstract fixed points to be computed in fewer iterations and also permits analysis of loops that may not terminate. In our setting, non-termination may reveal information about secret values. As such, we would like to reject queries that may be non-terminating.

We enforce this by not introducing a widening operator. Our abstract interpretation then has the property that it will not terminate if a loop in the query may be non-terminating (and, since it is an over-approximate analysis, it may also fail to terminate even for some terminating computations). We then reject all queries for which our analysis fails to terminate. Loops do not play a major role in any of our examples, and so this approach has proved sufficient so far. We leave for future work the development of a widening operator that soundly accounts for non-termination behavior.

Following standard abstract interpretation terminology, we will refer to $\mathcal{P}(\textbf{Dist})$ (sets of distributions) as the *concrete domain*, $\mathbb{P}$ as the *abstract domain*, and $\gamma_{\mathbb{P}} : \mathbb{P} \to \mathcal{P}(\textbf{Dist})$ as the *concretization function* for $\mathbb{P}$.

C. Abstract Semantics for $\mathbb{P}$

To support execution in the abstract domain just defined, we need to provide abstract implementations of the basic operations of assignment, conditioning, addition, and scaling used in the concrete semantics given in Figure 3. We will overload notation and use the same syntax for the abstract operators as we did for the concrete operators.

As we present each operation, we will also state the associated soundness theorem which shows that the abstract operation is an over-approximation of the concrete operation. Proofs are given in our technical report [27]. The abstract program semantics is then exactly the semantics from Figure 3, but making use of the abstract operations defined here, rather than the operations on distributions defined in Section III-B. We will write $\langle\!\langle S \rangle\!\rangle\, P$ to denote the result of executing S using the abstract semantics. The main soundness theorem we obtain is the following.

Theorem 6. *For all P, δ, if $\delta \in \gamma_{\mathbb{P}}(P)$ and $\langle\!\langle S \rangle\!\rangle\, P$ terminates, then $[\![S]\!]\delta$ terminates and $[\![S]\!]\delta \in \gamma_{\mathbb{P}}(\langle\!\langle S \rangle\!\rangle\, P)$.*

When we say $[\![S]\!]\delta$ terminates (or $\langle\!\langle S \rangle\!\rangle\, P$ terminates) we mean that only a finite number of loop unrollings are required to interpret the statement on a particular distribution (or probabilistic polyhedron). The precise definitions of termination are given in the technical report [27].

We now present the abstract operations.

1) Forget: We first describe the abstract forget operator $\mathrm{f}_y(P_1)$, which is used in implementing assignment. When we forget variable y, we collapse any states that are equivalent up to the value of y into a single state. To do this correctly, we must find an upper bound $\mathrm{h}_y^{\max}$ and a lower bound $\mathrm{h}_y^{\min}$ on the number of points that share the same value of other dimensions x (this may be visualized of as the min and max height of C_1 in the y dimension). Once these are obtained, we have that $\mathrm{f}_y(P_1) \stackrel{\text{def}}{=} P_2$ where the following hold of P_2.

$$\begin{aligned}
C_2 &= \mathrm{f}_y(C_1)\\
\mathrm{p}_2^{\min} &= \mathrm{p}_1^{\min} \cdot \max\left\{\mathrm{h}_y^{\min} - (\#(C_1) - \mathrm{s}_1^{\min}),\ 1\right\}\\
\mathrm{p}_2^{\max} &= \mathrm{p}_1^{\max} \cdot \min\left\{\mathrm{h}_y^{\max},\ \mathrm{s}_1^{\max}\right\}
\end{aligned}$$

$$\begin{array}{ll|ll}
\mathrm{s}_2^{\min} &= \lceil \mathrm{s}_1^{\min}/\mathrm{h}_y^{\max} \rceil & \mathrm{m}_2^{\min} &= \mathrm{m}_1^{\min}\\
\mathrm{s}_2^{\max} &= \min\left\{\#(\mathrm{f}_y(C_1)),\ \mathrm{s}_1^{\max}\right\} & \mathrm{m}_2^{\max} &= \mathrm{m}_1^{\max}
\end{array}$$

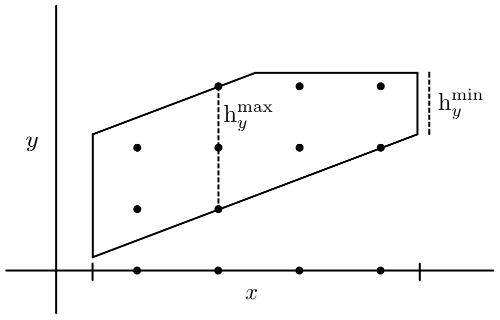


Figure 4. Example of a forget operation in the abstract domain $\mathbb{P}$. In this case, $\mathrm{h}_y^{\min} = 1$ and $\mathrm{h}_y^{\max} = 3$. Note that $\mathrm{h}_y^{\max}$ is precise while $\mathrm{h}_y^{\min}$ is an under-approximation. If $\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = 9$ then we have $\mathrm{s}_2^{\min} = 3$, $\mathrm{s}_2^{\max} = 4$, $\mathrm{p}_2^{\min} = \mathrm{p}_1^{\min} \cdot 1$, $\mathrm{p}_2^{\max} = \mathrm{p}_2^{\max} \cdot 4$.

Figure 4 gives an example of a forget operation and illustrates the quantities $\mathrm{h}_y^{\max}$ and $\mathrm{h}_y^{\min}$. If $C_1 = (B_1, V_1)$, the upper bound $\mathrm{h}_y^{\max}$ can be found by maximizing $y - y'$ subject to the constraints $B_1 \cup B_1[y'/y]$, where y' is a fresh variable and $B_1[y'/y]$ represents the set of constraints obtained by substituting y' for y in B_1. As our points are integer-valued, this is an integer linear programming problem (and can be solved by ILP solvers). A less precise upper bound can be found by simply taking the extent of the polyhedron C_1 along y, which is given by $\#(\pi_y(C_1))$.

For the lower bound, it is always sound to use $\mathrm{h}_y^{\min} = 1$, which is what our implementation does. A more precise estimate can be obtained by finding the vertex with minimal height along dimension y. Call this distance u. Since the shape is convex, all other points will have y height greater than or equal to u. We then find the smallest number of integer points that can be covered by a line segment of length u. This is given by $\lceil u \rceil - 1$. This value can be taken as $\mathrm{h}_y^{\min}$.

Since the forget operator is related to projection, we state soundness in terms of the projection operation on distributions. Note that $\mathit{fv}(\delta) \stackrel{\text{def}}{=} \mathit{domain}(\mathit{domain}(\delta))$, i.e., the domain of states to which δ assigns probability mass.

Lemma 7. *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta \upharpoonright (\mathit{fv}(\delta) - \{y\}) \in \gamma_{\mathbb{P}}(\mathrm{f}_y(P))$.*

We can define an abstract version of projection using forget:

Definition 8. Let $\mathrm{f}_{\{x_1, x_2, \ldots, x_n\}}(P) = \mathrm{f}_{\{x_2, \ldots, x_n\}}(\mathrm{f}_{x_1}(P))$. Then $P \upharpoonright V' = \mathrm{f}_{(\mathit{domain}(P) - V')}(P)$.

That is, in order to project onto the set of variables V', we forget all variables not in V'.

2) Assignment: We have two cases for abstract assignment. If $x := E$ is invertible, the result of the assignment $P_1\,[x \to E]$ is the probabilistic polyhedron P_2 such that $C_2 = C_1\,[x \to E]$ and all other components are unchanged.

If the assignment is not invertible, then information about the previous value of x is lost. In this case, we use the forget operation to project onto the other variables and then add a new constraint on x. Let $P_2 = \mathrm{f}_x(P_1)$ where $C_2 = (B_2, V_2)$.

Then $P_1[x \rightarrow E]$ is the probabilistic polyhedron P_3 with $C_3 = (B_2 \cup \{x = E\}, V_2 \cup \{x\})$ and all other components as in P_2.

Lemma 9. *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta[v \rightarrow E] \in \gamma_{\mathbb{P}}(P[v \rightarrow E])$.*

The soundness of assignment relies on the fact that our language of expressions does not include division. An invariant of our representation is that $\mathrm{s}^{\max} \leq \#(C)$. When E contains only multiplication and addition the above rules preserve this invariant; an E containing division would violate it. Division would collapse multiple points to one and so could be handled similarly to projection.

3) Plus: To soundly compute the effect of plus we need to determine the minimum and maximum number of points in the intersection that may be a support point for both P_1 and for P_2. We refer to these counts as the *pessimistic overlap* and *optimistic overlap*, respectively, and define them below.

Definition 10. Given two distributions δ_1, δ_2, we refer to the set of states that are in the support of both δ_1 and δ_2 as the *overlap* of δ_1, δ_2. The *pessimistic overlap* of P_1 and P_2, denoted $P_1 \frownie P_2$, is the cardinality of the smallest possible overlap for any distributions $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$. The *optimistic overlap* $P_1 \smiley P_2$ is the cardinality of the largest possible overlap. Formally, we define these as follows. $n_3 \overset{\text{def}}{=} \#(C_1 \sqcap_{\mathbb{C}} C_2)$, $n_1 \overset{\text{def}}{=} \#(C_1) - n_3$, and $n_2 \overset{\text{def}}{=} \#(C_2) - n_3$. Then

$$P_1 \frownie P_2 \overset{\text{def}}{=} \max\left\{(\mathrm{s}_1^{\min} - n_1) + (\mathrm{s}_2^{\min} - n_2) - n_3,\ 0\right\}$$
$$P_1 \smiley P_2 \overset{\text{def}}{=} \min\left\{\mathrm{s}_1^{\max}, \mathrm{s}_2^{\max}, n_3\right\}$$

We can now define abstract addition.

Definition 11. If not *iszero*(P_1) and not *iszero*(P_2) then $P_1 + P_2$ is the probabilistic polyhedron $P_3 = (C_3, \mathrm{s}_3^{\min}, \mathrm{s}_3^{\max}, \mathrm{p}_3^{\min}, \mathrm{p}_3^{\max})$ defined as follows.

$$\begin{aligned}
C_3 &= C_1 \sqcup_{\mathbb{C}} C_2 \\
\mathrm{p}_3^{\min} &= \begin{cases} \mathrm{p}_1^{\min} + \mathrm{p}_2^{\min} & \text{if } P_1 \frownie P_2 = \#(C_3) \\ \min\left\{\mathrm{p}_1^{\min}, \mathrm{p}_2^{\min}\right\} & \text{otherwise} \end{cases} \\
\mathrm{p}_3^{\max} &= \begin{cases} \mathrm{p}_1^{\max} + \mathrm{p}_2^{\max} & \text{if } P_1 \smiley P_2 > 0 \\ \max\left\{\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max}\right\} & \text{otherwise} \end{cases} \\
\mathrm{s}_3^{\min} &= \mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - P_1 \smiley P_2 \\
\mathrm{s}_3^{\max} &= \mathrm{s}_1^{\max} + \mathrm{s}_2^{\max} - P_1 \frownie P_2 \\
\mathrm{m}_3^{\min} &= \mathrm{m}_1^{\min} + \mathrm{m}_2^{\min} \quad \Big| \quad \mathrm{m}_3^{\max} = \mathrm{m}_1^{\max} + \mathrm{m}_2^{\max}
\end{aligned}$$

If *iszero*(P_1) then we define $P_1 + P_2$ as identical to P_2; if *iszero*(P_2), the sum is defined as identical to P_1.

Lemma 12. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ then $\delta_1 + \delta_2 \in \gamma_{\mathbb{P}}(P_1 + P_2)$.*

4) Product: When evaluating the product $P_3 = P_1 \times P_2$, we assume that the domains of P_1 and P_2 are disjoint, i.e., C_1 and C_2 refer to disjoint sets of variables. If $C_1 = (B_1, V_1)$ and $C_2 = (B_2, V_2)$, then the polyhedron $C_1 \times C_2 \overset{\text{def}}{=} (B_1 \cup B_2, V_1 \cup V_2)$ is the Cartesian product of C_1 and C_2 and contains all those states σ for which $\sigma \upharpoonright V_1 \in \gamma_{\mathbb{C}}(C_1)$ and $\sigma \upharpoonright V_2 \in \gamma_{\mathbb{C}}(C_2)$. Determining the remaining components is straightforward since P_1 and P_2 are disjoint.

$$C_3 = C_1 \times C_2$$
$$\begin{array}{rcl|rcl}
\mathrm{p}_3^{\min} &=& \mathrm{p}_1^{\min} \cdot \mathrm{p}_2^{\min} & \mathrm{p}_3^{\max} &=& \mathrm{p}_1^{\max} \cdot \mathrm{p}_2^{\max} \\
\mathrm{s}_3^{\min} &=& \mathrm{s}_1^{\min} \cdot \mathrm{s}_2^{\min} & \mathrm{s}_3^{\max} &=& \mathrm{s}_1^{\max} \cdot \mathrm{s}_2^{\max} \\
\mathrm{m}_3^{\min} &=& \mathrm{m}_1^{\min} \cdot \mathrm{m}_2^{\min} & \mathrm{m}_3^{\max} &=& \mathrm{m}_1^{\max} \cdot \mathrm{m}_2^{\max}
\end{array}$$

Lemma 13. *For all P_1, P_2 such that $fv(P_1) \cap fv(P_2) = \emptyset$, if $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ then $\delta_1 \times \delta_2 \in \gamma_{\mathbb{P}}(P_1 \times P_2)$.*

In our examples we often find it useful to express uniformly distributed data directly, rather than encoding it using pif. In particular, consider extending statements S to include the statement form uniform $x\ n_1\ n_2$ whose semantics is to define variable x as having values uniformly distributed between n_1 and n_2. Its semantics is as follows.

$$\langle\!\langle \mathsf{uniform}\ x\ n_1\ n_2 \rangle\!\rangle\, P_1 = \mathrm{f}_x(P_1) \times P_2$$

Here, P_2 has $\mathrm{p}_2^{\min} = \mathrm{p}_2^{\max} = \frac{1}{n_2 - n_1 + 1}$, $\mathrm{s}_2^{\min} = \mathrm{s}_2^{\max} = n_2 - n_1 + 1$, $\mathrm{m}_2^{\min} = \mathrm{m}_2^{\max} = 1$, and $C_2 = (\{x \geq n_1, x \leq n_2\}, \{x\})$.

We will say that the abstract semantics correspond to the concrete semantics of uniform defined similarly as follows.

$$[\![\mathsf{uniform}\ x\ n_1\ n_2]\!]\delta = (\delta \upharpoonright fv(\delta) - \{x\}) \times \delta_2$$

where $\delta_2 = (\lambda\sigma.\ \textbf{if}\ n_1 \leq \sigma(x) \leq n_2\ \textbf{then}\ \frac{1}{n_2 - n_1 + 1}\ \textbf{else}\ 0)$.

The soundness of the abstract semantics follows immediately from the soundness of forget and product.

5) Conditioning: Distribution conditioning for probabilistic polyhedra serves the same role as meet in the classic domain of polyhedra in that each is used to perform abstract evaluation of a conditional expression in its respective domain.

Definition 14. Consider the probabilistic polyhedron P_1 and Boolean expression B. Let $n, \overline{n}$ be such that $n = C_1 \# B$ and $\overline{n} = C_1 \# (\neg B)$. The value n is an over-approximation of the number of points in C_1 that satisfy the condition B and $\overline{n}$ is an over-approximation of the number of points in C_1 that do not satisfy B. Then $P_1 \mid B$ is the probabilistic polyhedron P_2 defined as follows.

$$\begin{array}{rl|rcl}
\mathrm{p}_2^{\min} &= \mathrm{p}_1^{\min} & \mathrm{s}_2^{\min} &=& \max\left\{\mathrm{s}_1^{\min} - \overline{n}, 0\right\} \\
\mathrm{p}_2^{\max} &= \mathrm{p}_1^{\max} & \mathrm{s}_2^{\max} &=& \min\left\{\mathrm{s}_1^{\max}, n\right\}
\end{array}$$
$$\begin{aligned}
\mathrm{m}_2^{\min} &= \max\left\{\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min},\ \mathrm{m}_1^{\min} - \mathrm{p}_1^{\max} \cdot \min\left\{\mathrm{s}_1^{\max}, \overline{n}\right\}\right\} \\
\mathrm{m}_2^{\max} &= \min\left\{\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max},\ \mathrm{m}_1^{\max} - \mathrm{p}_1^{\min} \cdot \max\left\{\mathrm{s}_1^{\min} - n, 0\right\}\right\} \\
C_2 &= \langle\!\langle B \rangle\!\rangle\, C_1
\end{aligned}$$

The maximal and minimal probability per point are unchanged, as conditioning simply retains points from the original distribution. To compute the minimal number of

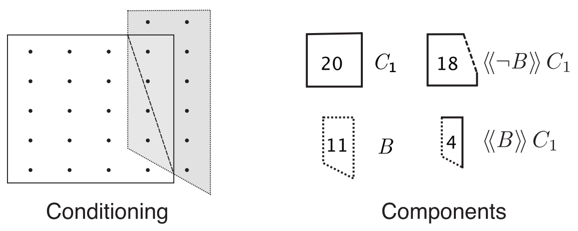

20
C_1
18
$\langle\!\langle \neg B \rangle\!\rangle\, C_1$
11
B
4
$\langle\!\langle B \rangle\!\rangle\, C_1$
Conditioning
Components

Figure 5. Example of distribution conditioning in the abstract domain $\mathbb{P}$.

points in P_2, we assume that as many points as possible from C_1 fall in the region satisfying $\neg B$. The maximal number of points is obtained by assuming that a maximal number of points fall within the region satisfying B.

The total mass calculations are more complicated. There are two possible approaches to computing $\mathrm{m}_2^{\min}$ and $\mathrm{m}_2^{\max}$. The bound $\mathrm{m}_2^{\min}$ can never be less than $\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min}$, and so we can always safely choose this as the value of $\mathrm{m}_2^{\min}$. Similarly, we can always choose $\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max}$ as the value of $\mathrm{m}_2^{\max}$. However, if $\mathrm{m}_1^{\min}$ and $\mathrm{m}_1^{\max}$ give good bounds on total mass (i.e., $\mathrm{m}_1^{\min}$ is much higher than $\mathrm{p}_1^{\min} \cdot \mathrm{s}_1^{\min}$ and dually for $\mathrm{m}_1^{\max}$), then it can be advantageous to reason starting from these bounds.

We can obtain a sound value for $\mathrm{m}_2^{\min}$ by considering the case where a maximal amount of mass from C_1 fails to satisfy B. To do this, we compute $\overline{n} = C_1 \# \neg B$, which provides an over-approximation of the number of points within C_1 but outside the area satisfying B. We bound $\overline{n}$ by $\mathrm{s}_1^{\max}$ and then assign each of these points maximal mass $\mathrm{p}_1^{\max}$, and subtract this from $\mathrm{m}_1^{\min}$, the previous lower bound on total mass.

By similar reasoning, we can compute $\mathrm{m}_2^{\max}$ by assuming a minimal amount of mass m is removed by conditioning, and subtracting m from $\mathrm{m}_1^{\max}$. This m is given by considering an under-approximation of the number of points falling outside the area of overlap between C_1 and B and assigning each point minimal mass as given by $\mathrm{p}_1^{\min}$. This m is given by $\max\bigl(\mathrm{s}_1^{\min} - n, 0\bigr)$.

Figure 5 demonstrates the components that affect the conditioning operation. The figure depicts the integer-valued points present in two polyhedra—one representing C_1 and the other representing B (shaded). As the set of points in C_1 satisfying B is convex, this region is precisely represented by $\langle\!\langle B \rangle\!\rangle\, C_1$. By contrast, the set of points in C_1 that satisfy $\neg B$ is not convex, and thus $\langle\!\langle \neg B \rangle\!\rangle\, C_1$ is an over-approximation. The icons beside the main image indicate which shapes correspond to which components and the numbers within the icons give the total count of points within those shapes.

Suppose the components of P_1 are as follows.

$$\begin{array}{lll} \mathrm{s}_1^{\min} = 19 & \mathrm{p}_1^{\min} = 0.01 & \mathrm{m}_1^{\min} = 0.85 \\ \mathrm{s}_1^{\max} = 20 & \mathrm{p}_1^{\max} = 0.05 & \mathrm{m}_1^{\max} = 0.9 \end{array}$$

Then $n = 4$ and $\overline{n} = 16$. Note that we have set $\overline{n}$ to be the number of points in the non-shaded region of Figure 5. This is more precise than the count given by $\#(\langle\!\langle B \rangle\!\rangle\, C)$, which would yield 18. This demonstrates why it is worthwhile to have a separate operation for counting points satisfying a boolean expression. These values of n and $\overline{n}$ give us the following for the first four numeric components of P_2.

$$\begin{array}{ll} \mathrm{s}_2^{\min} = \max(19 - 16, 0) = 3 & \mathrm{p}_2^{\min} = 0.01 \\ \mathrm{s}_2^{\max} = \min(20, 4) = 4 & \mathrm{p}_2^{\max} = 0.05 \end{array}$$

For the $\mathrm{m}_2^{\min}$ and $\mathrm{m}_2^{\max}$, we have the following for the method of calculation based on $\mathrm{p}_2^{\min/\max}$ and $\mathrm{s}_2^{\min/\max}$.

$$\mathrm{m}_2^{\min} = 0.01 \cdot 3 = 0.03 \qquad \mathrm{m}_2^{\max} = 0.05 \cdot 4 = 0.2$$

For the method of computation based on $\mathrm{m}_1^{\min/\max}$, we have

$$\begin{array}{lclcl} \mathrm{m}_2^{\min} & = & 0.85 - 0.05 \cdot 16 & = & 0.05 \\ \mathrm{m}_2^{\max} & = & 0.9 - 0.01 \cdot (19 - 4) & = & 0.75 \end{array}$$

In this case, the calculation based on subtracting from total mass provides a tighter estimate for $\mathrm{m}_2^{\min}$, while the method based on multiplying $\mathrm{p}_2^{\max}$ and $\mathrm{s}_2^{\max}$ is better for $\mathrm{m}_2^{\max}$.

Lemma 15. *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta | B \in \gamma_{\mathbb{P}}(P \mid B)$.*

6) Scalar Product: The scalar product is straightforward, as it just scales the mass per point and total mass.

Definition 16. Given a scalar p in $[0, 1]$, we write $p \cdot P_1$ for the probabilistic polyhedron P_2 specified below.

$$\begin{array}{lcl|lcl} \mathrm{s}_2^{\min} & = & \mathrm{s}_1^{\min} & \mathrm{p}_2^{\min} & = & p \cdot \mathrm{p}_1^{\min} \\ \mathrm{s}_2^{\max} & = & \mathrm{s}_1^{\max} & \mathrm{p}_2^{\max} & = & p \cdot \mathrm{p}_1^{\max} \\ \mathrm{m}_2^{\min} & = & p \cdot \mathrm{m}_1^{\min} & C_2 & = & C_1 \\ \mathrm{m}_2^{\max} & = & p \cdot \mathrm{m}_1^{\max} & & & \end{array}$$

Lemma 17. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ then $p \cdot \delta_1 \in \gamma_{\mathbb{P}}(p \cdot P_1)$.*

7) Normalization: If a probabilistic polyhedron P has $\mathrm{m}^{\min} = 1$ and $\mathrm{m}^{\max} = 1$ then it represents a normalized distribution. We define below an abstract counterpart to distribution normalization, capable of transforming an arbitrary probabilistic polyhedron into one containing only normalized distributions.

Definition 18. Whenever $\mathrm{m}_1^{\min} > 0$, we write normal(P_1) for the probabilistic polyhedron P_2 specified below.

$$\begin{array}{lcl|lcl} \mathrm{p}_2^{\min} & = & \mathrm{p}_1^{\min}/\mathrm{m}_1^{\max} & \mathrm{s}_2^{\min} & = & \mathrm{s}_1^{\min} \\ \mathrm{p}_2^{\max} & = & \mathrm{p}_1^{\max}/\mathrm{m}_1^{\min} & \mathrm{s}_2^{\max} & = & \mathrm{s}_1^{\max} \\ \mathrm{m}_2^{\min} & = & \mathrm{m}_2^{\max} = 1 & C_2 & = & C_1 \end{array}$$

When $\mathrm{m}_1^{\min} = 0$, we set $\mathrm{p}_2^{\max} = 1$. Note that if P_1 is the zero distribution then normal(P_1) is not defined.

Lemma 19. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and normal(δ_1) is defined, then normal$(\delta_1) \in \gamma_{\mathbb{P}}($normal$(P_1))$.*

D. Policy Evaluation

Here we show how to implement the threshold test given as Definition 3 using probabilistic polyhedra. To make the definition simpler, let us first introduce a bit of notation.

Notation 20. If P is a probabilistic polyhedron over variables V, and σ is a state over variables $V' \subseteq V$, then $P \mid \sigma \overset{\text{def}}{=} P \mid B$ where $B = \bigwedge_{x \in V'} x = \sigma(x)$.

Definition 21. Given some probabilistic polyhedron P_1 and statement S where $\langle\langle S \rangle\rangle\, P_1$ terminates, let $P_2 = \langle\langle S \rangle\rangle\, P_1$ and $P_3 = P_2 \upharpoonright L$. If, for every $\sigma_L \in \gamma_{\mathbb{C}}(C_3)$ with $\neg iszero(P_2 \mid \sigma_L)$, we have $P_4 = \text{normal}((P_2 \mid \sigma_L) \upharpoonright H)$ with $\text{p}_4^{\max} \leq t$, then we write $tsecure_t(S, P_1)$.

The computation of P_3 involves only abstract interpretation and projection, which are computable using the operations defined previously in this section. If we have a small number of outputs (as for the binary outputs considered in our examples), we can enumerate them and check $\neg iszero(P_2 \mid \sigma_L)$ for each output σ_L. When this holds (that is, the output is feasible), we compute P_4, which again simply involves the abstract operations defined previously. The final threshold check is then performed by comparing $\text{p}_4^{\max}$ to the probability threshold t.

Now we state the main soundness theorem for abstract interpretation using probabilistic polyhedra. This theorem states that the abstract interpretation just described can be used to soundly determine whether to accept a query.

Theorem 22. *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathbb{P}}(P_1)$ and $tsecure_t(S, P_1)$, then S is threshold secure for threshold t when evaluated with initial belief δ.*

VI. Powerset of Probabilistic Polyhedra

This section presents the $\mathcal{P}_n(\mathbb{P})$ domain, an extension of the $\mathbb{P}$ domain that abstractly represents a set of distributions as at most n probabilistic polyhedra, elements of $\mathbb{P}$.

Definition 23. A *probabilistic (polyhedral) set* Δ is a set of probabilistic polyhedra, or $\{P_i\}$ with each P_i over the same variables. We write $\mathcal{P}_n(\mathbb{P})$ for the domain of probabilistic polyhedral powersets composed of no more than n probabilistic polyhedra.

Each probabilistic polyhedron P is interpreted disjunctively: it characterizes one of many possible distributions. The probabilistic polyhedral set is interpreted additively. To define this idea precisely, we first define a lifting of $+$ to sets of distributions. Let D_1, D_2 be two sets of distributions. We then define addition as follows.

$$D_1 + D_2 = \{\delta_1 + \delta_2 \mid \delta_1 \in D_1 \wedge \delta_2 \in D_2\}$$

This operation is commutative and associative and thus we can use $\sum$ for summations without ambiguity as to order of operations. The concretization function for $\mathcal{P}_n(\mathbb{P})$ is then defined as:

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \overset{\text{def}}{=} \sum_{P \in \Delta} \gamma_{\mathbb{P}}(P)$$

We can characterize the condition of Δ containing only the zero distribution, written $iszero(\Delta)$, via the condition that all of the member probabilistic polyhedra are zero.

$$iszero(\Delta) \overset{\text{def}}{=} \bigwedge_{P \in \Delta} iszero(P)$$

A. Abstract Semantics for $\mathcal{P}_n(\mathbb{P})$

With a few exceptions, the abstract implementations of the basic operations for the powerset domain are extensions of operations defined on the base probabilistic polyhedra domain.

Theorem 24. *For all δ, S, Δ, if $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $\langle\langle S \rangle\rangle\, \Delta$ terminates, then $[\![S]\!]\delta$ terminates and $[\![S]\!]\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\langle\langle S \rangle\rangle\, \Delta)$.*

Definition 25. The *powerset simplification* transforms a set containing potentially more than n elements into one containing no more than n, for $n \geq 1$. The simplest approach involves repeated use of abstract plus in the base domain $\mathbb{P}$.

$$\lfloor \{P_i\}_{i=1}^m \rfloor_n \overset{\text{def}}{=} \begin{cases} \{P_i\}_{i=1}^m & \text{if } m \leq n \\ \lfloor \{P_i\}_{i=1}^{m-2} \cup \{P_{m-1} + P_m\} \rfloor_n & \text{otherwise} \end{cases}$$

Lemma 26. *$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\lfloor \Delta \rfloor_m)$ where $m \leq n$.*

Note that the order in which individual probabilistic polyhedra are simplified has no effect on soundness but may impact the precision of the resulting abstraction.

Many of the operations and lemmas for the powerset domain are simple liftings of the corresponding operations and lemmas for single probabilistic polyhedra. For these operations (operations 1-5 given below), we simply list the definition.

1) *Forget:* $\text{f}_y(\Delta) \overset{\text{def}}{=} \{\text{f}_y(P) \mid P \in \Delta\}$

2) *Project:* $\Delta \upharpoonright V \overset{\text{def}}{=} \{P \upharpoonright V \mid P \in \Delta\}$

3) *Conditioning:* $\Delta \mid B \overset{\text{def}}{=} \{P \mid B \mid P \in \Delta\}$

4) *Assignment:* $\Delta\,[x \to E] \overset{\text{def}}{=} \{P[x \to E] \mid P \in \Delta\}$

5) *Scalar product:* $p \cdot \Delta \overset{\text{def}}{=} \{p \cdot P \mid P \in \Delta\}$

6) *Product:* The product operation is only required for the special uniform statement and only applies to the product of a probabilistic set with a single probabilistic polyhedron. $\Delta \times P' \overset{\text{def}}{=} \{P \times P' \mid P \in \Delta\}$ (where we assume that $fv(\Delta) \cap P' = \emptyset$).

7) *Plus:* The abstract plus operation involves simplifying the combined contributions from two sets into one bounded set: $\Delta_1 + \Delta_2 \overset{\text{def}}{=} \lfloor \Delta_1 \cup \Delta_2 \rfloor_n$, whenever $\neg iszero(\Delta_1)$ and $\neg iszero(\Delta_2)$. Alternatively, if $iszero(\Delta_1)$ (or $iszero(\Delta_2)$) then $\Delta_1 + \Delta_2$ is defined to be identical to Δ_2 (or Δ_1).

8) Normalization: Since in the $\mathcal{P}_n(\mathbb{P})$ domain, the over(under) approximation of the total mass is not contained in any single probabilistic polyhedron, the normalization must scale each component of a set by the overall total. The minimum (maximum) mass of a probabilistic polyhedra set $\Delta = \{P_1, \ldots, P_n\}$ is defined as follows.

$$\mathrm{M}^{\min}(\Delta) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \mathrm{m}_i^{\min} \quad \Big| \quad \mathrm{M}^{\max}(\Delta) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \mathrm{m}_i^{\max}$$

Definition 27. The scaling of a probabilistic polyhedra P_1 by minimal total mass $\underline{m}$ and maximal total mass $\overline{m}$, written $\mathrm{normal}(P)(\underline{m}, \overline{m})$ is the probabilistic polyhedron P_2 defined as follows whenever $\underline{m} > 0$.

$$\begin{array}{rcl|rcl} \mathrm{p}_2^{\min} & = & \mathrm{p}_1^{\min}/\overline{m} & \mathrm{s}_2^{\min} & = & \mathrm{s}_1^{\min} \\ \mathrm{p}_2^{\max} & = & \mathrm{p}_1^{\max}/\underline{m} & \mathrm{s}_2^{\max} & = & \mathrm{s}_1^{\max} \\ \mathrm{m}_2^{\min} & = & \mathrm{m}_1^{\min}/\overline{m} & C_2 & = & C_1 \\ \mathrm{m}_2^{\max} & = & \mathrm{m}_1^{\max}/\underline{m} & & & \end{array}$$

Whenever $\underline{m} = 0$ the resulting P_2 is defined as above but with $\mathrm{p}_2^{\max} = 1$ and $\mathrm{m}_2^{\max} = 1$.

Normalizing a set of probabilistic polyhedra can be defined as follows

$$\mathrm{normal}(\Delta) \stackrel{\text{def}}{=} \left\{ \mathrm{normal}(P)(\mathrm{M}^{\min}(\Delta), \mathrm{M}^{\max}(\Delta)) \mid P \in \Delta \right\}$$

B. Policy Evaluation

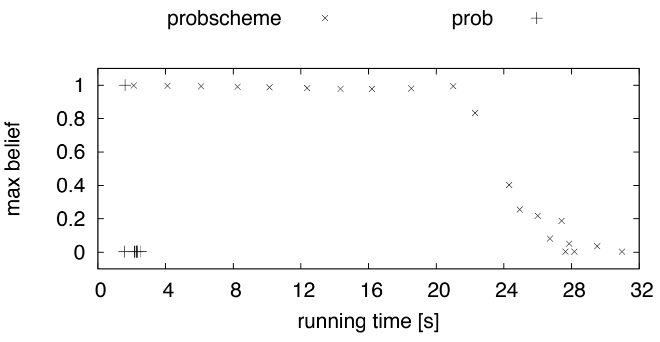
Determining the bound on the probability of any state represented by a single probabilistic polyhedron is as simple as checking the $\mathrm{p}^{\max}$ value in the normalized version of the probabilistic polyhedron. In the domain of probabilistic polyhedron sets, however, the situation is more complex, as polyhedra may overlap and thus a state's probability could involve multiple probabilistic polyhedra. A simple estimate of the bound can be computed by abstractly adding all the probabilistic polyhedra in the set, and using the $\mathrm{p}^{\max}$ value of the result. This is the approach we adopt in the implementation.

Lemma 28. *If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $P_1 = \sum_{P \in \Delta} P$ then $\max_\sigma \delta(\sigma) \leq \mathrm{p}_1^{\max}$.*

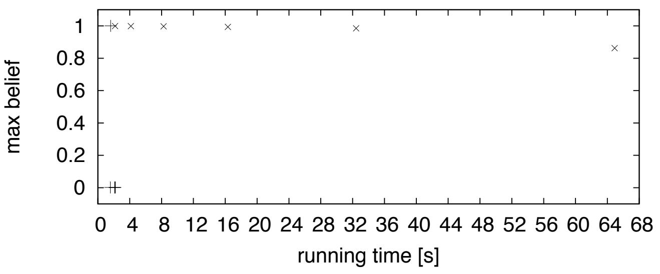
Notation 29. If Δ is a probabilistic polyhedron set over variables V, and σ is a state over variables $V' \subseteq V$, then $\Delta \mid \sigma \stackrel{\text{def}}{=} \Delta \mid B$ where $B = \bigwedge_{x \in V'} x = \sigma(x)$.

Definition 30. Given some probabilistic polyhedron Δ_1 and statement S where $\langle\!\langle S \rangle\!\rangle \Delta_1$ terminates, let $\Delta_2 = \langle\!\langle S \rangle\!\rangle \Delta_1$ and $\Delta_3 = \Delta_2 \upharpoonright L = \{P'_i\}$. If for every $\sigma_L \in \gamma_{\mathcal{P}(\mathbb{C})}(\{C'_i\})$ with $\neg \mathit{iszero}(\Delta_2 \mid \sigma_L)$ we have $\Delta_4 = \mathrm{normal}((\Delta_2 \mid \sigma_L) \upharpoonright H)$ and $P_4 = \sum_{P \in \Delta_4} P$ such that $\mathrm{p}_4^{\max} \leq t$, then we write $\mathit{tsecure}_t(S, \Delta_1)$.

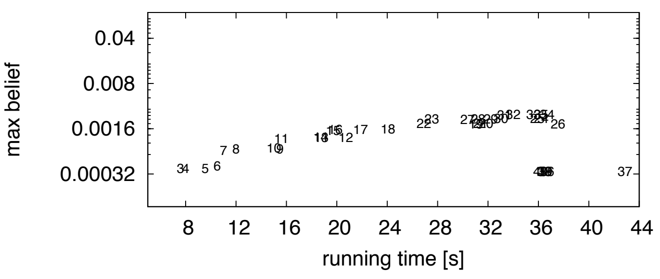
Below we state the main soundness theorem for abstract interpretation using probabilistic polyhedron sets. This theorem states that the abstract interpretation just described can be used to soundly determine whether to accept a query.



(a) birthday query (Example 1)



(b) birthday query (Example 1), larger state space



(c) special year query (Example 2)

Figure 6. Query evaluation comparison

Theorem 31. *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $\mathit{tsecure}_t(S, \Delta)$, then S is threshold secure for threshold t when evaluated with initial belief δ.*

VII. Implementation and experiments

We have implemented an interpreter for the core language based on the probabilistic polyhedra powerset domain. The base manipulations of polyhedra are done using the Parma Polyhedra Library [17]. Size calculations are done using the LattE lattice point counter [18]. LattE is also used for the integer linear programming problem involved in the abstract forget operation. The interpreter itself is written in OCaml.

We used our implementation to vet the query given in Example 1 (Section II) and compared its performance to an implementation based on Probabilistic Scheme [11], which is capable of sound probability estimation after partial enumeration.

Figure 6(a) illustrates the result when run on 2.5 GHz Intel

Core 2 Duo MacBook Pro, using OS X v10.5.8 with 4 GB of RAM. Each $\times$ plots Probscheme's maximum probability value (the y axis)—that is, the probability it assigns to the most likely secret state—when given a varying amount of time for sampling (the x axis). We can see the precision improves steadily until it reaches the exact value of 1/259 at around 30 seconds. Each $+$ plots our implementation's maximum probability value when given an increasing number of probabilistic polyhedra; with a polyhedral bound of 2 (or more), we obtain the exact value in less than 3 seconds. The advantage of our approach is more evident in Figure 6(b) where we use the same program but allow $byear$ to span 1910 to 2010 rather than 1956 to 1992. In this case ProbScheme makes little progress even after a minute, and eventually runs out of memory. Our approach, however, is unaffected by this larger state space and produces the exact maximum belief after around 3 seconds when using only 2 probabilistic polyhedra.

Figure 6(c) shows the result of our implementation assessing the special query (Example 2) with initial belief matching that following the first birthday query. Each plotted point is the number of polyhedra allowed. The result demonstrates that more complex queries, specifically ones with many disjunctions in their conditionals, not only slow our approach, but also reduce the precision of the maximum probability value. The example requires 36 polyhedra for exact calculations though as little as 3 produce probabilities near exact. The precision worsens as the number of polyhedra is increased until 36 are allowed. We conjecture this is due to an overly simple means of deciding which polyhedra to merge when performing abstract simplification; we plan to investigate this issue in future work.

VIII. Discussion and related work

Prior work aimed at controlling access to users' private data has focused on access control policies. For example, Persona [6] users can store personal data on distributed storage servers that use attribute-based encryption; only those parties that have the attribute keys for particular data items may see them. Our approach relaxes the access control model to offer more fine-grained information release policies by directly modeling an attacker's belief.

Others have considered how an adversary's knowledge of private data might be informed by a program's output. Clark, Hunt, and Malacaria [28] define a static analysis that bounds the secret information a straight-line program can leak in terms of equivalence relations between the inputs and outputs. Backes et al. [21] automate the synthesis of such equivalence relations and quantify leakage by computing the exact size of equivalence classes. Köpf and Rybalchenko [22] extend this approach, improving its scalability by using sampling to identify equivalence classes and using under- and over-approximation to obtain bounds on their size. Mu and Clark [29] present a similar analysis that uses over-approximation only. In all cases, the inferred equivalence classes can be used to compute entropy-based metrics of information leakage.

We differ from this work in two main ways. First, we implement a different security criterion. The most closely related metric is *vulnerability* V as proposed by Smith [20], which can be defined using our notation as follows:[2]

Definition 32. Let $\delta' = [\![S]\!]\delta$, where δ is the model of the querier's initial belief, and let $\overline{\delta_X} \overset{\text{def}}{=} \text{normal}(\delta \upharpoonright X)$. Then query S is *vulnerability threshold secure* iff for

$$V = \sum_{\sigma_L \in \mathit{support}(\delta'_L)} \overline{\delta'_L}(\sigma_L) \cdot \max_{\sigma_H \in \textbf{State}_H} \overline{(\delta'|\sigma_L)_H}(\sigma_H)$$

we have $V \leq t$ for some threshold t.

The above definition is an *expectation* over all possible outputs σ_L, so unlikely outputs have less influence. Our notion of threshold security (Definition 3) is stronger because it considers each output individually: if *any* output, however unlikely, would increase knowledge beyond the threshold, the query would be rejected. For example, recall the query from Example 1 where the secret data $bday$ is (assumed by the querier to be) uniformly distributed; call this query Q_1. According to Definition 32, the minimum acceptable threshold $t \geq V = 2/365 \approx 0.005$, whereas according to Definition 3, the minimum threshold is $t \geq 1/7 \approx 0.143$ which corresponds the equivalence class $260 \leq bday < 267$.

The other main difference is that we keep an on-line model of knowledge according to prior, actual query results, which increases our precision. To see the benefit consider performing query Q_1 followed by a query Q_2 which uses the code from Example 1 but has $today = 265$. With our system and $bday = 270$ the answer to Q_1 is False and with the revised belief the query Q_2 will be accepted as below threshold $t_d = 0.2$. If instead we had to model this pair of queries statically they would be rejected because (under the assumption of uniformity) the pair of outputs True,True is possible and implies $bday \in \{265, 266\}$ which would require $t_d \geq 0.5$. Our approach also inherits from the belief-based approach the ability to model a querier who is misinformed or incorrect, which can arise following the result of a probabilistic query (more on this below) or because of a change to the secret data between queries [9]. On the other hand, these advantages of our approach come at the cost of maintaining on-line belief models.

Our proposed abstract domains $\mathbb{P}$ and $\mathcal{P}_n(\mathbb{P})$ are useful beyond the application of belief-based threshold security; e.g., they could be used to model uncertainty off-line (as in the above work) rather than beliefs on-line, with the advantage that they are not limited to uniform distributions (as required by [21], [22]). Prior work on probabilistic abstract interpretation is insufficient for this purpose. For

[2]Smith actually proposes *min entropy*, which is $-log\ V$.

example, Monniaux [30] gives an abstract interpretation for probabilistic programs based on over-approximating probabilities. That work contains no treatment of distribution conditioning and normalization, which are crucial for belief-based information flow analysis. The use of under-approximations, needed to soundly handle conditioning and normalization, is unique to our approach.

McCamant and Ernst's FLOWCHECK tool [19] measures the information released by a particular execution. However, it measures information release in terms of *channel capacity*, rather than remaining uncertainty which is more appropriate for our setting. For example, FLOWCHECK would report a query that tries to guess a user's birthday leaks one bit regardless of whether the guess was successful, whereas the belief-based model (and the other models mentioned above) would consider a failing guess to convey very little information (much less than a bit), and a successful guess conveying quite a lot (much more than a bit).

To avoid reasoning directly about an adversary's knowledge, Dwork and colleagues proposed *differential privacy* [24]: a differentially private query over a database of individuals' records is a randomized function that produces roughly the same answer whether a particular individual's data is in the database or not. Thus, if the database curator is trustworthy, there is little reason for an individual to not supply his data. However, we prefer users to control access to their data as they like, rather than have to trust a curator.

In any case, it is difficult to see how to effectively adapt differential privacy, which was conceived for queries over many records, to queries over an individual's record, as in our setting. To see why, consider the birthday query from Example 1. Bob's birthday being/not being in the query range influences the output of the query only by 1 (assuming yes/no is 1/0). One could add an appropriate amount of (Laplacian) noise to the query answer to hide what the true answer was and make the query differentially private. However, this noise would be so large compared to the original range $\{0, 1\}$ that the query becomes essentially useless—the user would be receiving a birthday announcement most days.[3] By contrast, our approach permits answering queries exactly if the release of information is below the threshold. Moreover, there is no limit on the number of queries as long the information release remains bounded; differential privacy, in general, must impose an artificial limit (termed the *privacy budget*) because it does not reason about the information released.

Nevertheless, differential privacy is appealing, and it would be fruitful to consider how to apply its best attributes to our setting. Rastogi and Suciu [23] propose a property called *adversarial privacy* that suggests a way forward. Like our approach, adversarial privacy is defined in terms of a change in attacker knowledge. Roughly: a query's output on any database may increase an attacker's a priori belief $\delta(\sigma)$ about any state σ by at most ϵ for all $\delta \in D$ for some $D \in \mathcal{P}(\mathbf{Dist})$. Rastogi and Suciu show that, for a certain class D, adversarial privacy and differential privacy are equivalent, and by relaxing the choice of D one can smoothly trade off utility for privacy. We can take the reverse tack: by modeling a (larger) set of beliefs we can favor privacy over utility. Our abstractions $\mathbb{P}$ and $\mathcal{P}_n(\mathbb{P})$ already model sets of distributions, rather than a single distribution, so it remains interesting future work to exploit this representation toward increasing privacy.

Another important open question for our work is means to handle collusion. Following our motivating example in the Introduction, the user's privacy would be thwarted if he shared only his birth day with querier X and only his birth year with Y but then X and Y shared their information. A simple approach to preventing this would be to model adversary knowledge globally, effectively assuming that all queriers share their query results; doing so would prevent either X's or Y's query (whichever was last). This approach is akin to having a global privacy budget in differential privacy and, as there, obviously harms utility. Dealing with collusion is more problematic when using probabilistic queries, e.g., Example 2. This is because highly improbable results make a querier more uncertain, so combining querier knowledge can misrepresent individual queriers' beliefs. Roughly speaking, querier X could perform a query Q that misinforms the modeled global belief, but since querier Y's actual belief is not changed by the result of Q (since he did not actually see its result), he could submit Q' and learn more than allowed by the threshold. Disallowing probabilistic queries solves this problem but harms expressiveness. Another option is to more actively track a set of beliefs, as hinted at above.

IX. Conclusion

This paper has explored the idea of *knowledge-based security policies*: given a query over some secret data, that query should only be answered if doing so will not increase the querier's knowledge above a fixed threshold. We enforce knowledge-based policies by explicitly tracking a model of a querier's belief about secret data, represented as a probability distribution, and we deny any query that could increase knowledge above the threshold. Our denial criterion is independent of the actual secret, so denial does not leak information. We implement query analysis and belief tracking via abstract interpretation using novel domains of *probabilistic polyhedra* and *powersets of probabilistic polyhedra*. Compared to typical approaches to implementing belief revision, our implementation using this domain is more efficient and scales better.

Acknowledgments: The authors are grateful to Jeff Foster, Nataliya Guts, the anonymous referees, and especially Boris Köpf for their helpful comments on drafts of this paper.

[3] By our calculations, with privacy parameter $\epsilon = 0.1$ recommended by Dwork [24], the probability the query returns the correct result is approximately 0.5249.

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorised to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Support was also provided by National Science Foundation grants 0905419 and 0910530.

REFERENCES

[1] "Facebook developers," http://developers.facebook.com/, 2011, see the policy and docs/guides/canvas directories for privacy information.

[2] S. Guha, B. Cheng, and P. Francis, "Challenges in measuring online advertising systems," in *IMC*, 2010.

[3] "Statement of rights and responsibilities," http://www.facebook.com/terms.php, Oct. 2010.

[4] D. Worthington, "Myspace user data for sale," *PC World on-line*, Mar. 2010, http://www.pcworld.com/article/191716/myspace_user_data_for_sale.html.

[5] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam, "PrPl: a decentralized social networking infrastructure," in *1st International Workshop on Mobile Cloud Computing and Services: Social Networks and Beyond*, Jun. 2010, invited Paper.

[6] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *SIGCOMM*, 2009.

[7] P. Golle, "Revisiting the uniqueness of simple demographics in the us population," in *WPES*, 2006.

[8] "Facebook demographics and statistics report 2010," http://www.istrategylabs.com/2010/01/facebook-demographics-and-statistics-report-2010-145-growth-in-1-year/, 2010.

[9] M. R. Clarkson, A. C. Myers, and F. B. Schneider, "Quantifying information flow with beliefs," *J. Comput. Secur.*, vol. 17, no. 5, 2009.

[10] A. Pfeffer, "The design and implementation of IBAL: A general-purpose probabilistic language," in *Statistical Relational Learning*, L. Getoor and B. Taskar, Eds. MIT Press, 2007.

[11] A. Radul, "Report on the probabilistic language Scheme," in *DLS*, 2007.

[12] S. Park, F. Pfenning, and S. Thrun, "A probabilistic language based on sampling functions," *TOPLAS*, vol. 31, pp. 4:1–4:46, 2008.

[13] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," in *UAI*, 2008.

[14] O. Kiselyov and C.-C. Shan, "Embedded probabilistic programming," in *DSL*, 2009.

[15] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *POPL*, 1978.

[16] D. Monniaux, "Abstract interpretation of probabilistic semantics," in *Static Analysis Symposium (SAS'00)*, no. 1824, 200, pp. 322–339.

[17] "PPL: The Parma polyhedral library," http://www.cs.unipr.it/ppl/, 2011.

[18] J. Loera, D. Haws, R. Hemmecke, P. Huggins, J. Tauzer, and R. Yoshida, "Latte," http://www.math.ucdavis.edu/latte/, 2008.

[19] S. McCamant and M. D. Ernst, "Quantitative information flow as network flow capacity," in *PLDI*, 2008.

[20] G. Smith, "On the foundations of quantitative information flow," in *FOSSACS*, 2009.

[21] M. Backes, B. Köpf, and A. Rybalchenko, "Automatic discovery and quantification of information leaks," in *Security and Privacy*, 2009.

[22] B. Köpf and A. Rybalchenko, "Approximation and randomization for quantitative information-flow analysis," in *CSF*, 2010.

[23] V. Rastogi, M. Hay, G. Miklau, and D. Suciu, "Relationship privacy: output perturbation for queries with joins," in *PODS*, 2009.

[24] C. Dwork, "A firm foundation for private data analysis," *Commun. ACM*, vol. 54, pp. 86–95, Jan. 2011.

[25] R. Bagnara, P. M. Hill, and E. Zaffanella, "Widening operators for powerset domains," *Int. J. Softw. Tools Technol. Transf.*, vol. 8, pp. 449–466, 2006.

[26] C. Popeea and W. Chin, "Inferring disjunctive postconditions," in *In ASIAN CS Conference*, 2006.

[27] P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa, "Dynamic enforcement of knowledge-based security policies," University of Maryland Department of Computer Science, Tech. Rep. CS-TR-4978, Apr. 2011.

[28] D. Clark, S. Hunt, and P. Malacaria, "Quantitative information flow, relations and polymorphic types," *J. Log. and Comput.*, vol. 15, pp. 181–199, Apr. 2005.

[29] C. Mu and D. Clark, "An interval-based abstraction for quantifying information flow," *Electron. Notes Theor. Comput. Sci.*, vol. 253, pp. 119–141, November 2009.

[30] D. Monniaux, "Analyse de programmes probabilistes par interprétation abstraite," Thèse de doctorat, Université Paris IX Dauphine, 2001, résumé étendu en fran cais. Contents in English.