Symbolic Side-Channel Analysis for Probabilistic Programs

Pasquale Malacaria*, MHR. Khouzani*, Corina S. Păsăreanu†, Quoc-Sang Phan‡, Kasper Luckow†
*Queen Mary University of London, London E1 4FZ, UK
†Carnegie Mellon University, PA 15213, and NASA Ames Research Center, Moffett Field, CA 94035, USA
‡Fujitsu Laboratories of America, Sunnyvale, CA 94085, USA

***Abstract*—In this paper we describe symbolic side-channel analysis techniques for detecting and quantifying information leakage, given in terms of Shannon and min-entropy. Measuring the precise leakage is challenging due to the randomness and noise often present in program executions and side-channel observations. We account for this noise by introducing additional (symbolic) program inputs which are interpreted *probabilistically*, using symbolic execution with *parametrized* model counting. We also explore a sampling approach for increased scalability. In contrast to typical Monte Carlo techniques, our approach works by sampling symbolic paths, representing multiple concrete paths, and uses pruning to accelerate computation and guarantee convergence to the optimal results. A key novelty of our approach is to provide bounds on the leakage that are provably under- and over-approximating the exact leakage. We implemented the techniques in the Symbolic PathFinder tool and demonstrate them on Java programs.**

1. Introduction

Pervasive proliferation of computer systems has increased access to sensitive information, ranging from financial and medical records of individuals to trade and military secrets of corporations or countries. It has become critical to develop tools and techniques to ensure that software systems that manipulate sensitive data do so in a secure manner. The confidentiality of secret data is particularly hard to achieve if an attacker can observe non-functional characteristics of a program behavior, for instance by measuring execution times or memory usage. Side-channel attacks [7], [8], [20], [24] aim to recover secret program data based on this kind of observations. Side-channel attacks are of grave concern and they have been shown to pose serious security threats. For instance, they can recover cryptographic keys from well-known encryption/decryption algorithms [7] or derive user-sensitive data from common compression algorithms [20].

In this paper we describe symbolic analysis techniques for quantifying information leaks in side channels. A key obstacle in reasoning about side-channel attacks is the randomness and noise that are often present in program executions and side-channel measurements. This can come from “outside” the program, due to measurements made on noisy hardware platforms or over busy networks, but can also come from “inside” the program, due to explicit calls to `Random` methods in implementations of randomized algorithms (that are often used in security applications), or due to the thread scheduler or the garbage collector, which behave non-deterministically. Furthermore randomness can be introduced as a countermeasure against attacks, e.g. by adding random delays to a program to make it difficult for an adversary to infer secrets based on timing measurements. Thus, it is important to develop precise side channel analysis techniques to assess program leakage in the presence of random or noisy behavior and to determine if the implemented countermeasures are indeed effective.

To address this problem, we model both internal and external noise by creating additional (symbolic) inputs to the program. Conditions on these inputs are interpreted *probabilistically* using a probabilistic extension to symbolic execution [16]–[18]. The technique uses model counting over constraints collected with symbolic execution to compute probabilities of different program paths. In this paper, we adapt the technique to computing Shannon and min-entropy leakage in the presence of internal and external randomness (or noise). We further describe optimizations based on *symbolic projection* [1] to avoid explicit enumeration over secret values. Our analysis is parametrized by an observation model which allows us to obtain the side-channel measurements from the execution of program instructions.

To address the potential scalability issues associated with symbolic execution and model counting, we also explore partial exploration techniques that sample the symbolic program space to gradually collect symbolic paths. In contrast to typical Monte Carlo approaches, the techniques sample over the *symbolic paths* which represent multiple concrete paths. Furthermore, each symbolic path needs to be sampled only once and is subsequently pruned, accelerating the analysis. Our key contribution is to provide concrete (non-probabilistic) *lower* and *upper bounds* on the information leakage, for min-entropy and Shannon-Entropy leakage, based on the partial exploration analysis. The computed bounds hold for any partial exploration (hence they are “any-time”), improve with more symbolic samples and are guaranteed to converge to the exact value of leakage.

There are few existing tools that can cope with noisy observations. Statistical analysis tools, such as LeakWatch [9], are often imprecise and require a huge number of samples. These tools run the program multiple times and use the outcome of each execution to get an estimate of the information channel (joint probabilities) and any function of it, specifically, leakage values. In contrast to our symbolic approach, statistical approaches have the advantage that they can be used in a “black-box” fashion, where no knowledge of the program or noise distribution is assumed. However, due to the random nature of sampling, the leakage values provided by statistical approaches are only approximate, with no guarantee that they over- or under- approximate the real leakage, which may lead to many false results (positive or negative). Furthermore the accuracy measures such as the confidence intervals provided by e.g. LeakWatch are still probabilistic, and to get a reliable value for them one often needs a large amount of samples. In contrast, our symbolic analysis technique is exact and our estimation approach reports bounds that are provably under- and over-estimating the real leakage, while avoiding resampling of the same (symbolic) path.

We have implemented the exact and partial symbolic analyses in the Symbolic PathFinder tool [38]. We illustrate them in the context of Java applications where we show how to obtain tight bounds on the leakage in the presence of internal and/or external randomness. Our approach can be easily adapted to work with other symbolic execution tools, targeting other programming languages.

Contributions and roadmap.
The main contributions of this work are:

- Introducing a symbolic execution framework for the analysis of side channels in probabilistic programs (Sections 4 and 5);
- Combining this symbolic execution analysis with Barvinok parametrized model counting to optimize the computation (Section 6);
- Developing a symbolic sampling approach to scale up the computation, and proving guarantees in terms of converging concrete upper and lower bounds, both for Shannon and min-entropy leakage, for our symbolic sampling algorithms (Section 7);
- Demonstrating the applicability of these techniques to challenging programs (provided by DARPA) and exhibiting the merits of our techniques compared with tools that use standard Monte Carlo techniques; we use “Leakwatch” [9] for comparison (Section 8).

2. Motivating Example

We illustrate our analysis with the example in Figure 1. Assume that the sensitive data h is uniformly distributed over the domain $\mathcal{D} = \{0, 1, 2\}$, that is:

$$p(h=0) = p(h=1) = p(h=2) = \frac{1}{3}$$

The observable o, which can be for instance the execution time or the runtime power consumption, has three distinct values `obs1`, `obs2`, and `obs3`. The instance method `nextInt(2)` of `java.util.Random` also returns a uniformly distributed value from the set $\{0, 1\}$, and thus:

$$p(noise > 0) = p(\neg(noise > 0)) = \frac{1}{2}.$$

From the probability distribution of h and $noise$, constructing the joint probability matrix for (h, o) as in Figure 1 is straightforward. For example $(h = 0, o = \texttt{obs1})$ cannot happen, so its probability is 0; $(h = 1, o = \texttt{obs1})$ can only happen when $h = 1$ and $noise > 0$, hence its probability is $p(h = 1) \cdot p(noise > 0) = 1/3 \cdot 1/2 = 1/6$; $p(h = 2, o = \texttt{obs1}) = p(h = 2) = 1/3$ because when $h = 2$ the observable is `obs1` regardless of the noise.

```
java.util.Random rg = new
     Random();
int noise = rg.nextInt(2);
if(noise > 0){
  if(h > 0){
    obs1;
  } else {
    obs2;
  }
} else{
  if(h > 1){
    obs1;
  } else {
    obs3;
  }
}
```

		$p(o,h)$		
h	$p(h)$	obs1	obs2	obs3
0	1/3	0	1/6	1/6
1	1/3	1/6	0	1/6
2	1/3	1/3	0	0
$p(o)$		1/2	1/6	1/3

Figure 1: An example Java program and its corresponding joint probability matrix for (h, o).

An adversary can learn something about the secret h by observing o. From this joint probability matrix, one can quantify this leakage using security metrics such as Shannon entropy or (Rényi’s) min-entropy, as we will overview later.

3. Background: Symbolic Execution and Parametrized Model Counting

To analyze side channels, we use symbolic execution [22]. Symbolic execution is a well known program analysis technique that executes programs on symbolic inputs, representing multiple concrete inputs, and computes the effects of the program as symbolic expressions over the inputs. The result of the analysis is a set of symbolic paths, each with a path condition PC, which is a conjunction of constraints characterizing the inputs that follow that path. Path conditions are checked for satisfiability with off-the-shelf solvers such as Z3 [12] to ensure the symbolic execution follows only feasible paths. To deal with loops and recursion, typically a bound is put on the exploration depth.

We use parametrized model counting, specifically “Barvinok” [1], for computing the leakage of noisy side channels efficiently. Barvinok [1] provides an efficient procedure for model counting over linear integer arithmetic (LIA) constraints. A LIA constraint defines a multi-dimensional lattice bounded by a convex polytope [11]. The complexity of the algorithm does not depend on the actual size of the variable domains, making it suitable for the analysis of real programs (albeit limited to linear constraints).

In particular, we use Barvinok's `card` operation which computes the number of elements in a set, or the number of elements in the image of a domain element of a map [1, p.12]. The operation is performed exactly and symbolically and the result is a piecewise quasi-polynomial, i.e., a subdivision of one or more spaces, with a quasi-polynomial associated to each cell in the subdivision.

4. Analysis of Noisy Side Channels

Let $P(h)$ denote a program, where h is a high input (secret value) taken from the set $\mathcal{D}$. In this paper, we assume the high input (secret) is chosen uniformly from its space. We also assume that the public program inputs are fixed and can be hence treated as constants.

We briefly discuss relaxing the assumption of having a fixed low input in Section 6.3. Assuming a uniform distribution over the high values is compatible with practical situations, especially, in the context of cryptographic protocols, where the high value is the secret key. Also, if the distribution of the high can be selected, then it should be generally picked to be uniform, in order to maximize the uncertainty of an adversary. If the distribution cannot be picked, then the worst case for leakage (which pertains to "channel capacity") is still uniform distribution for the min-entropy leakage (though not for Shannon entropy).

Suppose an attacker tries to infer some information on the hidden secret h based on side channel measurements obtained when running the program. We assume that the attacker has full knowledge about the implementation of P. This assumption is justified as our goal is to give bounds on what side-channel attackers can, in principle, achieve.

Our setting is standard information theory [10]. With a slight abuse of notation, we use h and o to denote both the program secrets/output and the random variables representing them. For simplicity of presentation, we assume we have only one secret input; however all our results generalize to vectors of secrets in a straightforward way.

One way to compute the information leakage is based on Shannon information. Let the *observables* be the (side-channel) measurements computed for each path in the program and let $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$ denote these observations. Our analysis is parametrized by an *observation* function that gives these measurements for different kinds of side channels, e.g. execution time, number of allocated bytes, number of sent packets over a network, etc.

By definition, when considering systems whose only input is a confidential value h, Shannon leakage is the *mutual information* between the observable and the secret [29]:

$$\mathit{Leakage}^{\mathrm{S}} = \mathcal{I}(h; o) = \mathcal{H}(o) - \mathcal{H}(o|h), \tag{1}$$

where $\mathcal{H}(X)$ is classical Shannon entropy, measuring *uncertainty* about a random variable X. Let $p(o_i)$ be the probability of observing o_i (i.e. side-channel measurement in our case). Then, Shannon's entropy is defined as:

$$\mathcal{H}(o) = -\sum_{o_i \in \mathcal{O}} p(o_i) \log_2 p(o_i).$$

The second term in (1) is the conditional entropy of the observable given the input and is computed as follows:

$$\mathcal{H}(o|h) = \sum_{h_j \in \mathcal{D}} p(h_j) \mathcal{H}(o|h_j),$$

where $\mathcal{H}(o|h_j)$ is the Shannon entropy computed for the conditional probability of $p(o|h_j)$. $\mathcal{H}(o|h)$ can be thought of as the amount of uncertainty about the observable due to the effect of noise, and we will hence refer to it as "entropy of noise". If we consider a deterministic system and without any noise, then $\mathcal{H}(o|h) = 0$, since there is no uncertainty about the observable when the inputs are given. Thus the leakage is simply $\mathcal{H}(o)$. In the presence of noise, probabilistic scheduler or garbage collection, $\mathcal{H}(o|h) > 0$, so the leakage is smaller. Note that there are several equivalent ways to compute the leakage defined as Shannon mutual information. Namely:

$$\begin{aligned} \mathit{Leakage}^{\mathrm{S}} &= \mathcal{I}(h; o) = \mathcal{H}(o) - \mathcal{H}(o|h) = \\ &\mathcal{H}(o) + \mathcal{H}(h) - \mathcal{H}(h, o) = \mathcal{H}(h) - \mathcal{H}(h|o). \end{aligned}$$

where $\mathcal{H}(h, o)$ is the joint entropy of random variables h, o, given as: $\mathcal{H}(h, o) = -\sum_{i,j} p(h_j, o_i) \log_2 p(h_j, o_i)$.

Noise may come from outside the program, e.g. due to measurements made on a noisy hardware platform or over a busy network, but can also come from inside the program, e.g. due to explicit calls to `java.util.Random` methods. Further, noise can be due to the thread scheduler or the garbage collector, which behave non-deterministically. In this paper we internalize all these different kinds of noise. We introduce an additional input (or inputs) "noise", denoted by n, and we compute leakage of noisy programs $P(h)$ by analyzing programs of the form $P(h, n)$, using probabilistic symbolic execution [5], [16]. As we shall see, the analysis needs to be performed with care, and to treat these inputs differently when computing the conditional entropy.

As an example, consider again the code in Figure 1. We replace variable `noise` with an additional input with appropriate range (0..1 in this case) such that the probability of executing each branch of the condition involving this input is 50%, assuming a uniform input distribution. Similarly, for a multi-threaded program, we introduce symbolic inputs with appropriate ranges and conditions that execute with each context switch. Furthermore, noise in external measurements (e.g. made over a busy network) can be modeled by adding symbolic inputs and extra conditions that execute at the end of each program path. For the purpose of this paper, we assume that the operation modeling the probabilistic noise is given. In practice, this can be a challenging task requiring side information and extensive profiling of the system; this topic is, however, orthogonal to this work.

4.1. Min-entropy

We also consider the leakage expressed as *min-entropy* [40]. Min-entropy leakage measures how much the observations increase the probability of correctly guessing the secret in one try. Researchers have argued that to reason in terms of probability of guessing the secret is in many cases more natural and appropriate when quantifying leakage.

The leakage expressed as min-entropy (which we will refer to as min-leakage) is as follows:

$$Leakage^{\mathrm{M}} = \log_2 \left(\frac{\sum_i \max_j p(h_j, o_i)}{\max_j p(h_j)} \right). \qquad (2)$$

As a simple example justifying min-entropy, consider the following two probability distributions over the secret:

$$p_1 = (1/2, 1/2)\,, \quad p_2 = \left(1/2, 1/2^{1000}, \ldots, 1/2^{1000}\right).$$

These distributions have very different Shannon entropies (for p_1, it is 1 bit, while for p_2, it is 500.5 bits). However, they have the same min-entropy which capture that with probability $1/2$, the best guess is correct.

Since both Shannon and min-entropy formulations of leakage rely on joint probabilities, we will provide an efficient computation of the joint probabilities as one of the main objectives of our algorithms.

4.2. Observation models

The analysis is parametrized by an observation model, which provides measurements for the following:

- *execution time*: based on a model which maps each bytecode instruction to a timing observation, the execution time of a symbolic path is computed as the sum of timing observations of all instructions along that path.
- *memory usage*: by monitoring the Java Bytecode instructions that allocate memory (e.g., `NEW` and `NEWARRAY`), we can compute the number of heap objects allocated along a symbolic path. Further, we compute the memory allocated for the live heap objects along a path.
- *network and file communication*: the observable is the number of bytes written to a TCP socket or a file. In both cases, the writing is done via multiple invocations of the method `write` of `java.io.OutputStream`, which writes 1 byte to the stream. The listener monitors the bytecode instruction `INVOKEVIRTUAL` to check how many times `write` is called.

For cases that do not fall into the above categories, we developed another listener that monitors user-defined observables. A precise model of e.g. timing measurements needs to take into account low-level features of the target platform including processor architecture, pipelining, and branch prediction. For this paper, we assume that the observation model is given and consider building precise models out-of-scope. Furthermore, we do not consider cache side channels as they are out of scope for our Java project. However the tool that we use does have its own memory model that could be adapted for considering cache effects.

5. Computing Probabilities via Symbolic Execution and Model Counting

In this section, we describe the basics of how symbolic execution and model counting can be used to compute probabilities such as $p(o)$. In the next section, we present an efficient way to extend this method to compute joint probabilities $p(h, o)$ as well.

For $p(o)$, we adapt the procedure from previous work that studies noiseless side channels [37]. Specifically, we assume that the program under analysis, P, is modified to explicitly add the noise as an extra input. We perform a symbolic execution over the program $P(h, n)$ where we set the secret and noise inputs to fresh symbolic values (the inputs can be vectors) ranging over possibly very large but finite domains. We further assume that there are no infinite loops (loops with symbolic conditions will terminate due to the finite domain assumption).

The result of symbolic execution is a finite set of (feasible) symbolic paths $\pi_1, \pi_2, \ldots, \pi_n$ each with a path condition (PC_1, $PC_2, \ldots, PC_n$). Let $\mathrm{obs}(\pi_i)$ represent the side-channel measurement along that path according to the observation function. Thus we assume that each symbolic path can give only one observable, however, note that different paths may produce the same observable. Our work is done in the context of a project that targets side channels for Java programs but our results are applicable to other types of quantitative flow analysis where this assumption holds. To simplify the notation, we write $\mathrm{obs}(PC_i)$ to mean $\mathrm{obs}(\pi_i)$.

We group the path conditions that lead to the same observation o_i into *clauses*, i.e. $C_i \equiv \bigvee_{\mathrm{obs}(PC_j)=o_i} PC_j$. Each clause characterizes the set of secret and noise values that lead to the same observation. The procedure called **ComputeConstraints** is described below.

```
Procedure: ComputeConstraints($P(h, n), \mathrm{obs}(\cdot)$)
$\mathcal{O} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset$
$\mathcal{PC} \leftarrow SymbolicExecution(P(h, n))$
foreach $PC \in \mathcal{PC}$ do
    $\mathcal{O} \leftarrow \mathcal{O} \cup \{\mathrm{obs}(PC)\}$
foreach $o_i \in \mathcal{O}$ do
    $C_i \leftarrow \bigvee_{\mathrm{obs}(PC_j)=o_i} PC_j$
return $\mathcal{C}$
```

Assuming a uniform distribution over the secret, the probability of observing o_i is:

$$p(o_i) = \frac{\#(C_i)}{|\mathcal{D}| \cdot |\mathcal{D}_n|} = \frac{\sum\limits_{\mathrm{obs}(PC_j)=o_i} \#(PC_j)}{|\mathcal{D}| \cdot |\mathcal{D}_n|}.$$

Here $\#(c)$ denotes the number of solutions of c, i.e., possible values satisfying the constraint c, assuming that the secret and the noise values range over the domains $\mathcal{D}$ and $\mathcal{D}_n$, respectively. This count can be computed with an off-the-shelf model-counting procedure such as Barvinok [1].

We use $|\mathcal{D}|$ and $|\mathcal{D}_n|$ to denote the size of the secret and noise domains respectively. They can be large but finite.

Example. Consider the example introduced in Section 2. Running symbolic execution over the program (where both h and noise n are symbolic) results in four paths but only in three observables, since two paths lead to the same observation. As mentioned, we group the path conditions for the paths that lead to the same observation into a clause and we apply model counting over the clauses.

$$p(\texttt{obs1}) = \frac{\#(n{>}0 \wedge h{>}0) + \#(n{\leq}0 \wedge h{>}1)}{|\mathcal{D}| \cdot |\mathcal{D}_n|}$$
$$= \frac{|\{1,2\}| {\cdot} \#(n{>}0) + |\{2\}| {\cdot} \#(n{\leq}0)}{|\mathcal{D}| \cdot |\mathcal{D}_n|} = \frac{2{\cdot}1 + 1{\cdot}1}{3{\cdot}2} = \frac{1}{2}$$
$$p(\texttt{obs2}) = \frac{\#(n{>}0 \wedge h{\leq}0)}{|\mathcal{D}| \cdot |\mathcal{D}_n|} = \frac{|\{0\}|}{3} \cdot \frac{\#(n{>}0)}{|\mathcal{D}_n|} = \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6}$$
$$p(\texttt{obs3}) = \frac{\#(n{\leq}0 \wedge h{\leq}1)}{|\mathcal{D}| \cdot |\mathcal{D}_n|} = \frac{|\{0,1\}|}{3} \cdot \frac{\#(n{\leq}0)}{|\mathcal{D}_n|} = \frac{2}{3} \cdot \frac{1}{2} = \frac{1}{3}$$

6. Computing Joint Probabilities with Barvinok

Building on the discussion in the previous section, the joint probability can be computed as:

$$p(h_j, o_i) = \frac{\#(C_i \wedge h = h_j)}{|\mathcal{D}| \cdot |\mathcal{D}_n|}.$$

Recall that clause C_i is the disjunction of all the path conditions that lead to the same observable o_i. We propose to use Barvinok [1] to *decompose* the cardinality of the solution set for each C_i into its projections on the values of high input h and noise n. As we will demonstrate, this can be used to obtain efficient computations of the leakage.

To illustrate Barvinok's functionality, consider the following example, where a clause C is $(n > 0 \;\wedge\; h > 0) \vee (n \leq 0 \;\wedge\; h > 1)$, and the domains for n and h are $[0 \dots 2]$ and $[1 \dots 1000000]$ respectively. The result returned automatically by Barvinok is a function F_C as follows:

$$F_C(h) = \begin{cases} 2 & \textit{if } h = 1 \\ 3 & \textit{if } 2 \leq h \leq 1000000 \end{cases}$$

The result can be interpreted as follows. Given a choice for h, the number of solutions for C is $F_C(h)$. Thus, if $h = 1$, clause C has $F_C(1) = 2$ solutions. This is easy to verify since when $h = 1$ clause C becomes $(n > 0 \wedge 1 > 0) \vee (n \leq 0 \wedge 1 > 1)$ which simplifies to $(n > 0)$, which has 2 solutions. On the other hand, if $h = 2$, clause C has $F_C(2) = 3$ solutions. In fact, for any value of h such that $2 \leq h \leq 1000000$, there are three solutions satisfying C. Again this is easy to verify. For example, if $h = 2$, clause C becomes $(n > 0 \wedge 2 > 0) \vee (n \leq 0 \wedge 2 > 1)$ which simplifies to $(n > 0 \vee n \leq 0)$ which is satisfied by any value of noise. Another observation is that F_C gives the number of *noise* values that satisfy C for a particular value of h. Thus F_C can be seen as the decomposition of the solution space for $\#C$ with respect to h and n.

More generally, for any clause C_i we can use Barvinok's `card` operation to obtain a counting function $F_{C_i} : \mathcal{D} \to \mathbb{N}$, such that $F_{C_i}(h_j)$ gives the number of solutions for $(C_i \wedge h = h_j)$. We can then compute the joint probabilities as follows:

Proposition 1. $p(h_j, o_i) = \dfrac{F_{C_i}(h_j)}{|\mathcal{D}| \cdot |\mathcal{D}_n|}$.

The importance of this proposition is that the use of F_{C_i} is more efficient than the repeated invocation of a model counting procedure for computing $\#(C_i \wedge h = h_j)$ for each value of h_j in $\mathcal{D}$.

Combining the results of the previous and the current section, we can compute any leakage function based on the $p(h, o)$ and $p(o)$, including $\mathit{Leakage}^{\mathrm{S}}$ and $\mathit{Leakage}^{\mathrm{M}}$, but also other measures of interest like g-leakage [35].

6.1. Separable constraints

The computation of leakage based on Shannon entropy can be made more efficient for an interesting special case where the constraints on the noise n and high input h are *separable*. That is, each path condition PC can be written as $\phi_n \wedge \phi_h$, i.e., as a conjunction of constraints ϕ_n and ϕ_h, where ϕ_n is a constraint on noise variable and ϕ_h is a constraint on the secret. This can arise e.g. when the noise variables are not directly compared to secret values; this is a common case for noise that models multi-threading, imprecisions in hardware side-channel measurements, or defense mechanisms that add some noise to programs' computations. In our running example from Section 2, e.g., $PC_1 = n > 0 \wedge h > 0$. Thus: $\phi_n = (n > 0)$, $\phi_h = (h > 0)$.

For such cases, Barvinok's `card` operation generates a counting function of the following form:

$$F_{C_i}(h) = \begin{cases} N^i_{1,n} & h \models c^i_{1,h} \\ N^i_{2,n} & h \models c^i_{2,h} \\ \vdots & \\ N^i_{m,n} & h \models c^i_{m,h} \end{cases} \tag{3}$$

Here function F_{C_i} decomposes the solution space of $\#C_i$ in m components: for each component, the number of solutions is a *constant* number, denoted $N^i_{k,n}$ (for $k = 1, \dots, m$), when the secret h satisfies the corresponding condition, denoted $c^i_{k,h}$. The function $F_{C_i}(h)$ can be compactly represented by the set of pairs of $\langle N^i_{k,n}, c^i_{k,h} \rangle$ for $k = 1, \dots, m$. By construction, we have the following.

$$\#C_i = \sum_{k=1}^{m} N^i_{k,n} \cdot \#c^i_{k,h}.$$

Note that all $c^i_{k,h}$ are disjoint and their union defines all the h values that satisfy C_i; thus the conditions $c^i_{k,h}$ define a partition over the secrets satisfying C_i.

Computing Shannon Leakage. Notice that $\#(C_i \wedge h = h_j) = N^i_{k,n}$ if h_j is a solution to C_i (and further it is a solution of $c^i_{k,h}$), since the projection on h should have exactly one value, i.e. h_j, and it is 0 if h_j is not a solution to C_i. Further, the number of times $\#(C_i \wedge h = h_j) = N^i_{k,n}$ is exactly $\#c^i_{k,h}$. Thus we can use the piecewise decomposition for clause C_i to perform the following simplifications:

$$\begin{aligned}\sum_j \frac{\#(C_i \wedge h = h_j)}{|\mathcal{D}|\cdot|\mathcal{D}_n|}\cdot\log_2\frac{\#(C_i \wedge h = h_j)}{|\mathcal{D}|\cdot|\mathcal{D}_n|} = \\ \sum_{h\models c^i_{1,h}} \frac{N^i_{1,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}\cdot\log_2\frac{N^i_{1,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|} \\ + \sum_{h\models c^i_{2,h}} \frac{N^i_{2,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}\cdot\log_2\frac{N^i_{2,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|} \\ + \dots \\ + \sum_{h\models c^i_{m,h}} \frac{N^i_{m,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}\cdot\log_2\frac{N^i_{m,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|} \\ = \sum_{k=1}^{m} \#c^i_{k,h}\cdot\frac{N^i_{k,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}\cdot\log_2\frac{N^i_{k,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}.\end{aligned}$$

Thus, instead of iterating over all possible values of h (index j) we only need to iterate over the partitions over h (index k) as computed symbolically by Barvinok. It follows that:

Proposition 2. *For Shannon entropy, we can write:*

$$\mathcal{H}(h,o) = -\sum_i\sum_k \#c^i_{k,h}\cdot\frac{N^i_{k,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}\cdot\log_2\frac{N^i_{k,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}.$$

Thus, following $\mathcal{H}(o|h) = \mathcal{H}(h,o) - \mathcal{H}(h)$, we get:

$$\begin{aligned}\mathcal{H}(o|h) &= -\sum_{i,k}\#c^i_{k,h}\cdot\frac{N^i_{k,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}\cdot\log_2\frac{N^i_{k,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|} - \log_2|\mathcal{D}| \\ &= -\sum_{i,k}\#c^i_{k,h}\cdot\frac{N^i_{k,n}}{|\mathcal{D}|\cdot|\mathcal{D}_n|}\cdot\log_2\frac{N^i_{k,n}}{|\mathcal{D}_n|}.\end{aligned}$$

This proposition directly leads to Algorithm 2 below for an optimized way to compute the Shannon leakage.

Algorithm 2: ShannonLeakage

```
$h \leftarrow makeSymbolic('h')$
$\mathcal{C} \leftarrow$ ComputeConstraints$(P(h,n), \mathrm{obs}(\cdot))$
$\mathcal{H}_o \leftarrow 0$
$\mathcal{H}_{o|h} \leftarrow 0$
foreach $C_i \in \mathcal{C}$ do
    $p(o_i) = \frac{\#(C_i)}{|\mathcal{D}|\cdot|\mathcal{D}_n|}$
    $\mathcal{H}_o \leftarrow \mathcal{H}_o - p(o_i)\log_2(p(o_i))$
    $\mathcal{S}_i \leftarrow BarvH(C_i)$ /* BarvH: Barvinok 'card' operation, $F_{C_i}(h)$, as in (3) */
    foreach $\langle N_k, c_k\rangle \in \mathcal{S}_i$ do
        $\mathcal{H}_{o|h} \leftarrow \mathcal{H}_{o|h} - \frac{\#c_k}{|\mathcal{D}|}\cdot\frac{N_k}{|\mathcal{D}_n|}\log_2\left(\frac{N_k}{|\mathcal{D}_n|}\right)$
$Leakage^{\mathrm{S}} \leftarrow \mathcal{H}_o - \mathcal{H}_{o|h}$
return $Leakage^{\mathrm{S}}$
```

Example. Consider our running example again. Recall that there are four path conditions that lead to three observations. We therefore have three clauses:

$$\begin{aligned}C_1 &::= (h > 0 \wedge n > 0) \vee (h > 1 \wedge n \leq 0) \\ C_2 &::= h \leq 0 \wedge n > 0 \\ C_3 &::= h \leq 1 \wedge n \leq 0\end{aligned}$$

Running Barvinok on the three clauses gives the following:

$$\begin{aligned}F_{C_1}(h) &= \begin{cases} 2, & h = 2 \\ 1, & h = 1\end{cases} \\ F_{C_2}(h) &= 1, h = 0 \\ F_{C_3}(h) &= 1, 0 \leq h \leq 1\end{aligned}$$

Using our notations, $\mathcal{S}_1 = \{\langle 2, h = 2\rangle, \langle 1, h = 1\rangle\}$ etc. Thus, $\mathcal{H}(o|h) = -1\times 2/6\times\log_2 2/6 - 1\times 1/6\times\log_2 1/6 - 1\times 1/6\times\log_2 1/6 - 2\times 1/6\times\log_2 1/6 - \log_2 3 = 2/3$.

Computing min-entropy leakage. Min-entropy leakage can be computed in an efficient way in a similar manner, as provided in Algorithm 3. To see the logic behind the algorithm, note that within the inner loop, $N_k/\left(|\mathcal{D}|\cdot|\mathcal{D}_n|\right)$ for varying k, captures all the joint probabilities $p(h, o_i)$ for the given o_i, hence, the result of the inner loop (represented by variable max) will be $\max_j(p(h_j, o_i))$.

Algorithm 3: Min-entropy Leakage

```
$h \leftarrow makeSymbolic('h')$
$\mathcal{C} \leftarrow$ ComputeConstraint$(P(h,n), \mathrm{obs}(\cdot))$
$prob \leftarrow 0$
foreach $C_i \in \mathcal{C}$ do
    $max \leftarrow 0$
    $\mathcal{S}_i \leftarrow BarvH(C_i)$ /* BarvH: Barvinok 'card' operation, $F_{C_i}(h)$, as in (3) */
    foreach $\langle N_k, c_k\rangle \in \mathcal{S}_i$ do
        if $max < \frac{N_k}{|\mathcal{D}|\cdot|\mathcal{D}_n|}$ then
            $max \leftarrow \frac{N_k}{|\mathcal{D}|\cdot|\mathcal{D}_n|}$
    $prob \leftarrow prob + max$
$Leakage^{\mathrm{M}} \leftarrow \log_2(prob\cdot|\mathcal{D}|)$
return $Leakage^{\mathrm{M}}$
```

For our running example, we can compute $prob = 2/6 + 1/6 + 1/6 = 2/3$ giving $Leakage^{\mathrm{M}} = \log_2(2/3\times 3) = 1$.

6.2. Control-flow Side Channels

We discuss a special case of interest, where each observable is associated with a single path condition and the constraints on noise and secret are separable. This describes the so called "control-flow side channels" [2], [34], [39]. In such cases, the attacker can identify the executed path by observing the trace of all program counter values, or as shown recently, all memory access locations during execution.

As before, we can decompose each path condition PC_i as: $\phi_n^i \wedge \phi_h^i$. Since ϕ_n and ϕ_h are separated, we simply have $\#PC_i = \#\phi_n^i \cdot \#\phi_h^i$. Hence:

$$p(o_i) = \frac{\#PC_i}{|\mathcal{D}| \cdot |\mathcal{D}_n|} = \frac{\#\phi_h^i \cdot \#\phi_n^i}{|\mathcal{D}| \cdot |\mathcal{D}_n|}.$$

We can use this observation and the calculations for $\mathcal{H}(o|h)$ from the previous sections to simplify the leakage formula:

Proposition 3. *When each observable is associated to a single path condition and the constraints on secret and noise are separable, we have:*

$$Leakage^{\mathrm{S}} = -\sum_i p(o_i) \log_2 \frac{\#\phi_h^i}{|\mathcal{D}|}.$$

The corresponding algorithms are omitted for brevity.

6.3. Leakage Computation in the Presence of Both High and Low inputs

We now discuss the case when the program depends not only on the confidential input h but also on the public input l. In this case the leakage is defined using conditional mutual information [29], i.e.:

$$\mathcal{I}(h; o|l) = \mathcal{H}(o|l) - \mathcal{H}(o|h, l)$$

As before, in the case of deterministic systems the term $\mathcal{H}(o|h, l)$ is 0 and the above reduces to $\mathcal{H}(o|l)$. When we fix a low input, it can be seen as a constant within the program, and we can then see the program as a system that only depends on the high input, for which we can use the leakage formulas we have presented in the previous sections.

We can also treat the public input l symbolically and apply parametrized model counting and numerical optimization to find the low value that maximizes the leakage, in a manner similar to [36] (that work does not consider noise). The technical development of such an approach is beyond the scope of this paper, and we leave it for future work.

7. Symbolic Sampling With Path Pruning

The leakage computations that we have presented can be computationally expensive as they involve exploring all the symbolic program paths (up to a user specified bound) and performing expensive (parametrized) model counting over the collected path conditions. We describe here alternative *sampling techniques* to scale up the computation of leakage.

Instead of an exhaustive symbolic execution, we perform sampling over the symbolic paths, as follows: A sample is obtained by performing one symbolic execution run; whenever a condition depending on a symbolic value is encountered, the decision of exploring either the *then* or the *else* branch is taken randomly, breaking the tie arbitrarily, resulting in one symbolic path through the program.

Note that instead of doing Monte Carlo sampling as done in previous approaches [9], [19] we perform sampling over the *symbolic paths*. Each symbolic path represents multiple concrete paths all following the same control flow in the program. It is not necessary to sample along the same path multiple times, since repeated sampling will not result in new information. We can therefore *prune* the explored paths, reducing the search space and accelerating the analysis.

The result of this analysis, after taking multiple samples, is a *subset* of the symbolic paths of the program, over which we can apply the leakage computation. Our goal is to be able to compute *bounds (lower and upper bounds)* on the actual leakage of the program given this partial symbolic exploration. The approach that we propose is an *anytime* analysis, that is, we would like bounds that hold for any partial exploration of the program, i.e., yield valid bounds whenever the estimation program is halted.

But first, we make a simpler observation: since the analysis with pruning always explores new paths, and since we assume a finite number of paths, it follows that the search converges to an exhaustive analysis and that precise program leakage will be computed eventually.

Proposition 4 (Convergence of symbolic sampling with pruning)**.** *If pruning is enabled, then all the symbolic program paths are guaranteed to be sampled yielding the precise leakage computation for the program.*

7.1. Anytime Bounds for Leakage

As we mentioned, even with symbolic execution and Barvinok counting, the process of computing leakage for a practical program can be long. Especially, each time a loop is encountered in the program, it can be unwounded a number of times during symbolic execution resulting in an impractical number of paths to explore. Symbolic sampling with path-pruning will explore only a subset of the symbolic program paths, leading to partial learning of the (side) channels of the program. In this section, we provide both upper and lower bounds for the leakage given any partial sampling of the channel. Both of the bounds monotonically converge to the actual value of leakage as more sample paths are examined. These bounds are any-time bounds, in the following sense: they are guaranteed that at any time that their computation is halted, the bounds are valid, and the leakage is sandwiched between them. Furthermore, under some conditions, our bounds are "tight", in that, no better bound exist given the samples so far. In other words, there exist programs that could generate the samples learned so far (are consistent with them) and have a leakage that is exactly equal to one of the bounds.

Let $\widehat{p}(o_i|h_j)$ be the estimated conditional probability of observable o_i given the high value h_j after some (but not all) symbolic paths are sampled. Specifically, $\widehat{p}(o_i|h_j) = \#(C_i(h_j))/|\mathcal{D}_n|$, where $\#(C_i(h_j))$ represents the number of noise values that satisfy the clause C_i for the given h_j based on the collected symbolic paths. Compared to the actual value of $p(o_i|h_j)$ we have:

$$\widehat{p}(o_i|h_j) \leq p(o_i|h_j), \quad \forall i, j.$$

This is because the remaining symbolic paths may only add to the value of conditional probabilities that are seen so far, and can never subtract from it.

Note that some of the high inputs as well as some observables may not have been part of any samples yet. Let the observables that are seen so far be $o_1, \ldots, o_k$ out of $o_1, \ldots, o_m$, where $k \leq m$. The value of m (the size of the space of the observables) may not even be known in advance (before sampling is complete, unlike the size of the space of the high inputs $|\mathcal{D}|$, which is known even before sampling begins). Also, keep in mind that $\widehat{p}(o_i|h_j)$ may not yet constitute proper conditional probabilities, that is, we can have: $\sum_{i=1}^{k} \widehat{p}(o_i|h_j) < 1$. This is because for a given h_j, there may be some observables that are not sampled yet, or there may be some contributions to the sampled observables that are yet to be accounted for. Recall that the (prior) distribution over the high values is assumed to be uniform. We can also define $\widehat{p}(h_j, o_i)$ and $\widehat{p}(o_i)$ to respectively be the learned joint probability of (h_j, o_i), and the probability of output o_i, by symbolic sampling so far. That is:

$$\widehat{p}(h_j, o_i) = \frac{1}{|\mathcal{D}|} \widehat{p}(o_i|h_j),$$

$$\widehat{p}(o_i) = \frac{1}{|\mathcal{D}|} \sum_j \widehat{p}(o_i|h_j).$$

Moreover, let $q(h_j)$ be defined as the following:

$$q(h_j) = \frac{1}{|\mathcal{D}|} \left(1 - \sum_i \widehat{p}(o_i|h_j) \right). \tag{4}$$

That is, $q(h_j)$ can be thought of as the "unaccounted for" probability for high input h_j that is to be learned through future samples. Note that the value of $q(h_j)$ can be measured given the samples so far. Initially, $q(h_j)$ is at $1/|\mathcal{D}|$, and as more paths are sampled, its value goes down to zero. Finally, let Q be the total unexplored probability. That is:

$$Q = \sum_j q(h_j) = 1 - \sum_{i,j} \widehat{p}(h_j, o_i) = 1 - \sum_i \widehat{p}(o_i). \tag{5}$$

Given this background, we can now present our bounds.

7.2. Upper bounds for side-channel leakage

Proposition 5. *Let $\widehat{p}(o|h)$ be the learned values of the (side)-channel of the program resulting from symbolic sampling. Then U_{S} as given below is a tight monotonically decreasing upper-bound for the Shannon leakage:*

$$U_{\mathrm{S}} = \log_2 |\mathcal{D}| - \widehat{\mathcal{H}}(h|o), \quad \textit{where:}$$

$$\widehat{\mathcal{H}}(h|o) = -\sum_{i,j} \widehat{p}(h_j, o_i) \log_2 \frac{\widehat{p}(h_j, o_i)}{\widehat{p}(o_i)}.$$

Similarly, the following is a tight monotonically decreasing upper-bound for the min-entropy leakage:

$$U_{\mathrm{M}} = \log_2 \left(|\mathcal{D}| \sum_i \max_j \widehat{p}(h_j, o_i) + |\mathcal{D}| \cdot Q \right).$$

Note that before any sampling, the value of the upper-bound (for both Shannon and min-entropy) is simply $\log_2 |\mathcal{D}|$, i.e., $\mathcal{H}(h)$, which corresponds to a program that completely reveals the value of high.

Algorithms 4 and 5 show the implementation of the proposition for Shannon and min-entropy, respectively.

Algorithm 4: UpperBoundShannon$(P(h,n), \mathrm{obs}(\cdot))$

```
$\mathcal{O} \leftarrow \emptyset$, $Q \leftarrow 1$, $U \leftarrow \log_2(|\mathcal{D}|)$
$t \leftarrow 0$                /* counter for path samples */
while $Q > 0$ do
    $t \leftarrow t + 1$
    $PC \leftarrow$ sample a new symbolic path
    $o \leftarrow \mathrm{obs}(PC)$
    $\pi \leftarrow \frac{\#PC}{|\mathcal{D}| \cdot |\mathcal{D}_n|}$
    $Q \leftarrow Q - \pi$
    if $o \notin \mathcal{O}$ then                /* new observable */
        $\langle N, \phi_h \rangle \leftarrow BarvH(PC)$
        $\widehat{p}[o] \leftarrow \pi$
        $u[o] \leftarrow -\pi \cdot \log_2(\#\phi_h)$
          /* $u[o_i] = p(o_i)\mathcal{H}(h|o_i)$ in $U = \sum_i u[o_i]$ */
        $U[t] \leftarrow U[t-1] + u[o]$
        $\mathcal{O} \leftarrow \mathcal{O} \cup o$
        $C[o] \leftarrow PC$
    else  /* observable already encountered */
        $U[t] \leftarrow U[t-1] - u[o]$      /* subtracting
          the old $u[o_i] = \widehat{p}(o_i)\widehat{\mathcal{H}}(h|o_i)$ as the only
          term in $U = \sum_i u[o_i]$ that changes. */
        $\widehat{p}[o] \leftarrow \widehat{p}[o] + \pi$
        $C[o] \leftarrow C[o] \vee PC$
        $u[o] \leftarrow 0$     /* computing the new $u[o]$ */
        foreach $\langle N_j, c_j \rangle \in BarvH(C[o])$ do
            $p_j \leftarrow \frac{N_j}{|\mathcal{D}| \cdot |\mathcal{D}_n|}$
            $u[o] \leftarrow u[o] + \#c_j \cdot p_j \cdot \log_2 \left(\frac{p_j}{\widehat{p}[o]}\right)$
        $U[t] \leftarrow U[t] + u[o]$
    Prune the symbolic path $PC$
return $U$
```

Proof: We provide the proof for a generalized class of leakages that includes Shannon and min-entropy as special cases. Assume the expression for the posterior entropy $\mathcal{H}(h|o)$ can be written as follows:

$$\mathcal{H}(h|o) = \eta\Big(\sum_{i:p(o_i)>0} p(o_i) F\left(p(h|o_i)\right) \Big), \tag{6}$$

where η is an $\mathbb{R} \to \mathbb{R}$ function, and F is a scalar function over the space of probability distributions and one of the following two cases holds:

$$\eta\text{: increasing, and } F\text{: concave; or} \tag{7a}$$

$$\eta\text{: decreasing, and } F\text{: convex.} \tag{7b}$$

Shannon follows (6) by choosing η to be the identity function $\eta(x) = x$, and $F\left(p(h|o_i)\right) =$

Algorithm 5: UpperBoundMinEnt$(P(h,n), \mathrm{obs}(\cdot))$

```
$\mathcal{O} \leftarrow \emptyset$, $Q \leftarrow 1$, $U \leftarrow 1$
$t \leftarrow 0$
while $Q > 0$ do
    $t \leftarrow t + 1$
    $PC \leftarrow$ sample a new symbolic path
    $o \leftarrow \mathrm{obs}(PC)$
    $Q_{\text{new}} \leftarrow Q - \frac{\#PC}{|\mathcal{D}| \cdot |\mathcal{D}_n|}$
    if $o \notin \mathcal{O}$ then
        $\langle N, \phi_h \rangle \leftarrow BarvH(PC)$
        $u[o] \leftarrow \frac{N}{|\mathcal{D}| \cdot |\mathcal{D}_n|}$    /* $u[o_i] = \max_{h_j} \widehat{p}(h_j, o_i)$ */
        $U[t] \leftarrow U[t-1] + u[o] + (Q_{\text{new}} - Q)$
        $\mathcal{O} \leftarrow \mathcal{O} \cup o$
        $C[o] \leftarrow PC$
    else
        $U[t] \leftarrow U[t-1] - u[o]$
        $C[o] \leftarrow C[o] \vee PC$
        foreach $\langle N_j, c_j \rangle \in BarvH(C[o])$ do
            $u[o] \leftarrow \max\left\{u[o], \frac{N_j}{|\mathcal{D}| \cdot |\mathcal{D}_n|}\right\}$
        $U[t] \leftarrow U[t] + u[o] + (Q_{\text{new}} - Q)$
    Prune the symbolic path $PC$
return $\log_2(|\mathcal{D}| \cdot U)$
```

$-\sum_j p(h_j|o_i)\log_2(p(h_j|o_i))$, which corresponds to (7a). Posterior min-entropy, $-\log_2\left(\sum_i p(o_i)\max_j(p(h_j|o_i))\right)$, can be expressed by taking $\eta(x) = -\log_2(x)$ and $F(p(h|o_i)) = \max_j(p(h_j|o_i))$, which complies with (7b).

We can rewrite (6) as $\mathcal{H}(h|o) = \eta(K(p(h,o)))$, by introducing the function K as follows:

$$K(p(h,o)) = \sum_{i:p(o_i)>0} p(o_i) F(p(h|o_i)). \tag{8}$$

The domain of K can be extended to include any non-negative $p(h_j, o_i)$, by naturally defining:

$$p(o_i) = \sum_j p(h_j, o_i), \quad p(h|o_i) = (p(h_j|o_i))_j = \left(\frac{p(h_j,o_i)}{p(o_i)}\right)_j$$

By $(p(h_j|o_i))_j$, we mean the $\mathbb{R}^{+|\mathcal{D}|}$ vector whose entries are $p(h_j|o_i)$ and $j = 1, \ldots, |\mathcal{D}|$. Note that whenever $p(o_i) > 0$, the vector $p(h|o_i)$ as defined above is still a legitimate probability distribution for h, because $p(h_j|o_i) \geq 0\ \forall j$, and $\sum_j p(h_j|o_i) = 1$. Hence, $p(h|o_i)$ is still in the domain of F. We prove the proposition using the following lemma:

Lemma 1. *For* (7a)*, K is "super-additive", i.e., for any $p_1(h,o)$ and $p_2(h,o)$ on the same space $\mathbb{R}^{+|\mathcal{D}|\,|\mathcal{O}|}$, we have:*

$$K(p_1(h,o) + p_2(h,o)) \geq K(p_1(h,o)) + K(p_2(h,o)).$$

For the case (7b), we have "sub-additivity", i.e., the inequality of the lemma holds in the opposite direction.

Proof: We provide the proof only for case (7a) for brevity. First, we show that for $\alpha > 0$, $K(\alpha p(h,o)) = \alpha K(p(h,o))$: Transforming $p(h,o)$ to $\alpha p(h,o)$ transforms $p(o_i) = \sum_j p(h_j, o_i)$ to $\sum_j \alpha p(h_j, o_i) = \alpha p(o_i)$. On the other hand, $p(h_j|o_i) = p(h_j, o_i)/p(o_i)$ transforms to $\alpha p(h_j, o_i)/(\alpha p(o_i))$, which is the same as $p(h_i|o_i)$. Hence:

$$K(\alpha p(h,o)) = \sum_{i:\alpha p(o_i)>0} \alpha p(o_i) F(p(h|o_i)) = \alpha K(p(h,o)) \tag{9}$$

Next, following Proposition 1 in [21], we note that K is concave, i.e., $\forall p_1(h,o)$ and $p_2(h,o)$ on the same space $\mathbb{R}^{+|\mathcal{D}|\,|\mathcal{O}|}$, and $\forall \lambda \in [0,1]$, we have:

$$K(\lambda p_1(h,o) + (1-\lambda)p_2(h,o)) \geq \lambda K(p_1(h,o)) + (1-\lambda)K(p_2(h,o)).$$

Now, consider $p_1(h,o)$ and $p_2(h,o)$ in $\mathbb{R}^{+|\mathcal{D}|\,|\mathcal{O}|}$. Take a $\lambda \in (0,1)$. Following concavity of K:

$$K\left(\lambda \cdot \frac{p_1(h,o)}{\lambda} + (1-\lambda) \cdot \frac{p_2(h,o)}{1-\lambda}\right) \geq \lambda \cdot K\left(\frac{p_1(h,o)}{\lambda}\right) + (1-\lambda) \cdot K\left(\frac{p_2(h,o)}{1-\lambda}\right)$$

The left-hand is simply $K(p_1(h,o) + p_2(h,o))$. Using (9), the right-hand simplifies to $K(p_1(h,o)) + K(p_2(h,o))$. □

Resuming with the proof of Proposition 5, let us define $q(h,o)$ as the difference between the true joint probabilities and the learned ones so far. That is, let:

$$q(h,o) := p(h,o) - \widehat{p}(h,o)$$

where the difference is performed element-wise (i.e., $\forall i, j$). Since any partial symbolic sampling provides an under-estimation of the joint probabilities, we have $\widehat{p}(h,o) \leq p(h,o)$, or equivalently $q(h,o) \geq 0$. Now, because we can write $p(h,o) = \widehat{p}(h,o) + q(h,o)$, where both $\widehat{p}(h,o)$ and $q(h,o)$ are element-wise non-negative, Lemma 1 implies:

$$K(p(h,o)) \geq K(\widehat{p}(h,o)) + K(q(h,o)) \tag{10}$$

Referring to the definition of K in (8), for Shannon, since η is the identity, this inequality reduces to:

$$\mathcal{H}(h|o) \geq \widehat{\mathcal{H}}(h|o)) + \mathcal{H}_q(h|o)$$

where $\mathcal{H}_q(h|o) = -\sum_{h,o} q(h,o)\log_2\left(q(h,o)/\sum_h q(h,o)\right)$, which is the familiar posterior Shannon entropy where $q(h,o) \geq 0$ is used in place of $p(h,o)$. It can be easily shown (using e.g., Jensen's inequality) that $\mathcal{H}_q(h|o) \geq 0$. Referring to the definition of leakage, we thus have:

$$\mathit{Leakage}^{\mathrm{S}} = \mathcal{H}(h) - \mathcal{H}(h|o) \leq \mathcal{H}(h) - \widehat{\mathcal{H}}(h|o)$$

Note that $\mathcal{H}(h)$ is simply the entropy of the high value, which for uniform distribution is simply $\log_2(|\mathcal{D}|)$.

For min-entropy, the conditional entropy can be written as $-\log_2(K(p(h,o)))$, where $K(p(h,o)) = \sum_i \max_j p(h_j, o_i)$, which is a convex function (hence, we have case (7b)). Therefore, inequality (10) holds in the opposite direction:

$$\sum_i \max_j p(h_j, o_i) \leq \sum_i \max_j \widehat{p}(h_j, o_i) + \sum_i \max_j q(h_j, o_i)$$

Since, $-\log_2(x)$ is a decreasing function over $x > 0$, and leakage is the difference between prior entropy, $\log_2 |\mathcal{D}|$, and posterior entropy, we get:

$$\textit{Leakage}^{\mathrm{M}} \leq \log_2 \left(|\mathcal{D}| \sum_i \max_j \widehat{p}(h_j, o_i) + |\mathcal{D}| \sum_i \max_j q(h_j, o_i) \right)$$

The claim in the proposition follows by noting that the term $\sum_i \max_j q(h_j, o_i)$ can be bounded above by $\sum_i \sum_j q(h_j, o_i) = \sum_j q(h_j) = Q$.

Showing that these upper-bounds are tight is by constructing a (side)-channel whose leakage is exactly the bound. Such channel is constructed by assigning the remaining probability of each high value, $q(h_i, o_j)$, to a distinct new observable. Intuitively, this is the maximally leaking channel among all possible channels that are consistent with the partially learned channel. □

Note that the tightness property in Proposition 5 is under the assumption that nothing is known in advance about the size of the observable space $|\mathcal{O}|$. If $|\mathcal{O}|$ or an upper-bound on it is known before sampling, then we may no longer have tightness, and our upper-bounds can be improved. For instance, suppose it is known (or can be argued) that $|\mathcal{O}| \leq M$ for a constant M where $M < |\mathcal{D}|$. Then we know leakage is less than $\log_2 M$ which is better than the upper-bound in the proposition before any sampling, i.e., $\log_2 |\mathcal{D}|$.

If sampling is done over the symbolic paths obtained by initializing the high input one at a time, i.e., scanning the high values concretely while using model counting on the symbolic noise, then this upper-bound can be improved (proof omitted for brevity):

Proposition 6. *Suppose the number of distinct possible (side channel) observables is known to be capped at M. When sampling is done by concrete scanning of the high values and model counting on the symbolic noise, the following is also an upper-bound for Shannon Leakage:*

$$U'_{\mathrm{S}} = \min\{\widehat{\mathcal{H}}(o) - Q \log_2 \frac{Q}{M}, \log_2 M\} - \widehat{\mathcal{H}}(h, o)$$
$$+(1-Q)\log_2 |\mathcal{D}|, \quad \textit{where:}$$
$$\widehat{\mathcal{H}}(o) = -\sum_i \widehat{p}(o_i) \log_2 \widehat{p}(o_i), \quad \textit{and}$$
$$\widehat{\mathcal{H}}(h, o) = -\sum_{i,j} \widehat{p}(h_j, o_i) \log_2 \widehat{p}(h_j, o_i).$$

7.3. Lower bounds for side-channel leakage

Proposition 7. *Let $\widehat{p}(o|h)$ be the learned values of the (side)-channel of the program resulting from symbolic sampling so far. Then L_{S} as given below is an any-time lower-bound for the Shannon leakage:*

$$L_{\mathrm{S}} = \log_2 |\mathcal{D}| - \widehat{\mathcal{H}}(h|o) + \sum_j q(h_j) \log_2 q(h_j)$$
$$+ (1-Q)\log_2(1-Q)$$

where $\widehat{\mathcal{H}}(h|o)$ is as defined in Proposition 7, and $q(h_j)$ and Q are as defined in (4) *and* (5)*, respectively. Moreover, the following is a monotonically non-decreasing lower-bound for the min-entropy leakage:*

$$L_{\mathrm{M}} = \max \left(\log_2 \left(|\mathcal{D}| \sum_i \max_j \widehat{p}(h_j, o_i) \right), 0 \right)$$

At the beginning of sampling, the value of the leakage's lower bound for both entropies is zero, which corresponds to the possibility that the program satisfies non-interference. Both lower-bounds converge to the real value by the end of sampling, for min-entropy, monotonically so.

Proof: For min-entropy, the proof is straightforward by referring to the definition of its leakage in (2) for a uniform distribution on the secrets, and noting two points: first, the leakage can never be negative, and second, that the estimated joint probabilities $\widehat{p}(h, o)$ is always a lower bound for (and can only increase to) the true joint probabilities $p(h, o)$ using our symbolic sampling.

The rest of the proof concerns Shannon. Recall:

$$\mathcal{H}(h) - \mathcal{H}(h|o) = \mathcal{H}(h) + \sum_{i,j} p(h_j, o_i) \log_2 \left(\frac{p(h_j, o_i)}{p(o_i)} \right)$$

in which $p(h, o)$ is the true joint probability, and we have: $p(h, o) = \widehat{p}(h, o) + q(h, o)$, where $q(h, o) \geq 0$. Hence $-\mathcal{H}(h|o)$ can be written as:

$$\sum_{i,j} (\widehat{p}(h_j, o_i) + q(h_j, o_i)) \log_2 \left(\frac{\widehat{p}(h_j, o_i) + q(h_j, o_i)}{\widehat{p}(o_i) + q(o_i)} \right)$$

Consider the function $f(t) = t \log_2(t)$. $f(t)$ is convex and $f(0) = 0$, therefore, it is super-additive. Hence, we have:

$$\sum_{i,j} (\widehat{p}(h_j, o_i) + q(h_j, o_i)) \log_2 \left(\frac{\widehat{p}(h_j, o_i) + q(h_j, o_i)}{\widehat{p}(o_i) + q(o_i)} \right) \geq$$
$$\sum_{i,j} \widehat{p}(h_j, o_i) \log_2 \left(\frac{\widehat{p}(h_j, o_i)}{\widehat{p}(o_i) + q(o_i)} \right) +$$
$$\sum_{i,j} q(h_j, o_i) \log_2 \left(\frac{q(h_j, o_i)}{\widehat{p}(o_i) + q(o_i)} \right) \quad (11)$$

In what follows, we find a tight lower-bound for the right hand side (and hence, for $-\mathcal{H}(h|o)$) by casting it as an optimization problem and solve it in closed-form:

$$\min_{q(h,o)} \Big\{ \sum_{i,j} \widehat{p}(h_j, o_i) \log_2 \left(\frac{\widehat{p}(h_j, o_i)}{\widehat{p}(o_i) + q(o_i)} \right)$$
$$+ \sum_{i,j} q(h_j, o_i) \log_2 \left(\frac{q(h_j, o_i)}{\widehat{p}(o_i) + q(o_i)} \right) \Big\} \quad (12a)$$

subject to:

$$q(h_j, o_i) \geq 0, \forall i, \forall j \quad \& \quad \sum_i q(h_j, o_i) = q(h_j), \forall j \quad \text{(12b)}$$

Lemma 2. *The following is a closed-form solution for* (12)*:*

$$q^*(h_j, o_i) = q(h_j) \frac{\widehat{p}(o_i)}{\sum_{i'} \widehat{p}(o_{i'})} \qquad (13)$$

The lemma follows by checking that the proposed solution satisfies the Karush-Khun-Tucker (KKT) conditions for the convex optimization (12). Details are omitted for brevity.

Next, we will replace the minimizing argument given by the lemma into the objective function in (12a) as a lower-bound on $-\mathcal{H}(h|o)$. First, (13) gives:

$$q^*(o_i) = \sum_j q^*(h_j, o_i) = \sum_j q(h_j) \frac{\widehat{p}(o_i)}{\sum_{i'} \widehat{p}(o_{i'})} = \widehat{p}(o_i) \frac{Q}{1-Q}$$

Hence, (12a) becomes:

$$\begin{aligned}
&\sum_{i,j} \widehat{p}(h_j, o_i) \log_2 \left(\frac{\widehat{p}(h_j, o_i)}{\widehat{p}(o_i) + \widehat{p}(o_i) \frac{Q}{1-Q}} \right) \\
&\qquad + \sum_{i,j} q(h_j) \frac{\widehat{p}(o_i)}{1-Q} \log_2 \left(\frac{q(h_j) \frac{\widehat{p}(o_i)}{1-Q}}{\widehat{p}(o_i) + \widehat{p}(o_i) \frac{Q}{1-Q}} \right) \\
&= \sum_{i,j} \widehat{p}(h_j, o_i) \log_2 \left(\frac{\widehat{p}(h_j, o_i)(1-Q)}{\widehat{p}(o_i)} \right) \\
&\qquad + \sum_{i,j} q(h_j) \frac{\widehat{p}(o_i)}{1-Q} \log_2 (q(h_j)) \\
&= \sum_{i,j} \widehat{p}(h_j, o_i) \log_2 \left(\frac{\widehat{p}(h_j, o_i)}{\widehat{p}(o_i)} \right) + \sum_{i,j} \widehat{p}(h_j, o_i) \log_2 (1-Q) \\
&\qquad + \sum_{i,j} q(h_j) \frac{\widehat{p}(o_i)}{1-Q} \log_2 (q(h_j)) \\
&= \sum_{i,j} \widehat{p}(h_j, o_i) \log_2 \left(\frac{\widehat{p}(h_j, o_i)}{\widehat{p}(o_i)} \right) + (1-Q) \log_2 (1-Q) \\
&\qquad + \sum_j q(h_j) \log_2 (q(h_j))
\end{aligned}$$

Referring to the definition of $\widehat{\mathcal{H}}(h|o)$ in Proposition 5, the first term in the last equation above is $-\widehat{\mathcal{H}}(h|o)$. In short: $-\mathcal{H}(h|o) \geq -\widehat{\mathcal{H}}(h|o) + (1-Q)\log_2 (1-Q) + \sum_j q(h_j) \log_2 q(h_j)$ which yields:

$$\begin{aligned}
\mathit{Leakage}^{\mathrm{S}} \geq \mathcal{H}(h) - \widehat{\mathcal{H}}(h|o) + (1-Q)\log_2 (1-Q) + \\
\sum_j q(h_j) \log_2 q(h_j)
\end{aligned}$$

For uniform prior over high values, $\mathcal{H}(h) = \log_2(|\mathcal{D}|)$. □

While for a general sampling, these bounds may not be tight in general, we have the following tightness result:

Proposition 8. *When sampling is done by concrete scanning of the high values and model counting on the symbolic noise, then the lower bounds in Proposition 7 become tight.*

Again, by tightness, we mean the program under consideration can achieve the lower-bound as its leakage, so it is impossible to improve the lower-bounds.

Proof: For Shannon, the inequality of (11) is satisfied with equality when for all i, j, either $\hat{p}(h_j, o_i)$ or $q(h_j, o_i)$ is zero. This is exactly the case when sampling is done by concretely scanning the high value while keeping the noise symbolic. For min-entropy follows by noting that the lower-bound in Proposition (7) is exactly achieved for the legitimate channel described in (13). □

Note that when using the above per-input sampling, the value of $q(h_j)$ is either zero (for a sampled input), or $p(h) = 1/|\mathcal{D}|$ (for an unexplored input). Hence, the term $\sum_j q(h_j) \log_2 q(h_j)$ in the expression of Shannon's lower-bound simply becomes $-Q \log_2 |\mathcal{D}|$. When using path sampling with both high and noise symbolic, this term, although computable, is not as straightforward to keep track of. One way to go around this is to use a simpler (albeit looser) lower-bound by noticing that:

$$\sum_j q(h_j) \log_2 q(h_j) \geq Q \log_2(Q/D)$$

with equality occurring only when $q(h_j) = (\sum_j q(h_j))/|\mathcal{D}| = Q/|\mathcal{D}|$ for all j. Using this inequality gives the overall lower bound as:

$$L'_{\mathrm{S}} = \log_2 |\mathcal{D}| - \widehat{\mathcal{H}}(h|o) + Q \log_2 \frac{Q}{|\mathcal{D}|} + (1-Q) \log_2 (1-Q) \quad (14)$$

Hence, compromising on the quality of the lower-bound, we have something that is now easier to compute.

Confidence Intervals. Our bounds implies that the true value of the leakage is within the "error" window of $[L_{\mathrm{x}}, U_{\mathrm{x}}]$, $x \in \{\mathrm{S}, \mathrm{M}\}$, which goes to zero. The value of this error window can be used to stop the sampling once it falls below a desired threshold. Hence, similar to previous work on probabilistic analysis [16] but in the strong notion of this paper, we can define a *confidence interval* $CI_{\mathrm{x}} = [L_{\mathrm{x}}, U_{\mathrm{x}}]$. Our confidence interval is non-probabilistic, that is, the actual leakage is guaranteed to be within it, and, in case of tightness, it cannot be improved.

8. Implementation and Experiments

We implemented the described techniques in the Symbolic PathFinder (SPF) tool, using listeners to monitor the execution of symbolic paths, collect the constraints, perform model counting and sampling. We evaluated them on the following examples, initially provided by DARPA as part of STAC engagements.

1) `CheckSecret`, a password-checker that compares a "guess" with an internally stored password, "secret" (see Figure 2), encoded as numbers. The program goes recursively over the elements of an array `t` of random boolean variables. The size refers to the length of array `t`. Our techniques discovered a timing side-channel caused by the delays in the code.
2) `CRIME`, an instance of the CRIME attack ("Compression Ratio Info-leak Made Easy") [15]. We analyzed procedure `compress(high.concat(low))`,

```
boolean checkSecret(int guess, int[] t){
  return recur(guess, t, t.length - 1);
}

boolean recur(int guess, int[] t, int index){
  if(index == 0 && t[index] == 1){
    if(guess <= secret){
      Thread.sleep(10);
    }
    return guess == secret;
  } else if(t[index] == 1){
    return recur(guess, t, index -1);
  }
  if(guess <= secret){
    Thread.sleep(10);
  }
  return guess == secret;
}
```

Figure 2: The `CheckSecret` example. One of the example programs that we investigate in Section 8.

which implements string compression using the Lempel-Ziv (LZ77) algorithm. We studied both a deterministic and a randomized version: with a probability $p = 3/10$, $k = 1$ bytes are appended to the compressed string before it is sent. The size here refers to the the number of characters of high and low strings (i.e., input lengths, which we kept equal for simplicity); our techniques discovered a space side channel, that reveals the secret through the size of the compressed input.

3) `LawDB`, a network service application that manages information about law enforcement personnel. Each employee's data is referenced with a unique employee ID. Standard users are provided operations such as SEARCH and VIEW to manage information about employees except for the ones working on confidential activities. IDs of this particular group of employees are restricted. The requests handling non-confidential records are randomly delayed. In this case, "size" indicates the number of IDs in the database. Our techniques reported a timing channel in the process of handling a SEARCH request that is caused by additional computation, leading to slower response, when there are restricted IDs within the requested search range.

4) `Collab`, a complex application implementing a network service that provides event scheduling. Users can insert events into the `Collab` calendar. The events are organized by their ID in a re-balancing tree structure. In addition to regular users, `Collab` also has special *auditors* that can insert auditing events for other users. It is not possible for regular users to obtain auditing events directly. The application performs logging during the maintenance of the balanced tree. A random number determines how many lines are written to the log file. Our analysis identified both a timing and a space channel that reveals the IDs related to a user's audits. This is due to the fact that when each node exceeds a certain number of IDs, the node is split into two, which results in some extra logging, hence taking additional time.

8.1. Full Exploration of Symbolic Paths

We first evaluated our exact approach. The results are summarized in Table 1. For each example, we report the number of paths (#PC), the Shannon and the min-entropy leakages, along with the corresponding execution time. To illustrate the scaling behavior of our approach, we examined our examples for different size configurations.

For the `CRIME` example, we ran experiments on both the deterministic and the randomized versions (columns marked as `CRIME` and noisy `CRIME`): the results show that, as expected, the randomized version has smaller leakage for both Shannon and min-entropy measures.

8.2. Symbolic Path Sampling

We also experimented with our sampling approach on these examples. Figure 3 plots the lower and upper bounds of Shannon leakage for `CRIME` as more path samples (shown in percentage on x axis) are analysed. The plot shows how all the bounds converge to the exact leakage. Furthermore, if at any time the exploration is stopped, the real value of leakage is guaranteed to be between the lower and upper-bounds. A similar pattern was observed for all the other examples (see e.g. Figure 5 for `CheckSecret`).

We observe that many samples are needed for the tight bound U_{S} tight* (given by Proposition 5) to get close to the real value. If a bound on the number of observables is known, then a trivial upper-bound on the leakage is the "capacity" bound, i.e., $\log_2 |\mathcal{O}|$. This can be improved using Proposition 6 to an upper-bound that monotonically decreases to the exact leakage (marked as U_{S} using $|\mathcal{O}|$). The L_{S} tight is as provided by Proposition 7, whose tightness comes from Proposition 8. Finally, "L_{S} simple" is as per (14), which is of course not as good as the "L_{S} tight".

The plots for min-entropy are illustrated in e.g. Figures 6 (see also Fig. 4 for `CheckSecret`). The plots show how all the bounds converge to the exact leakage faster than the Shannon leakage case. Note also that our results can be used for side channel *detection*. For example, if the lower bound for min-entropy leakage is greater than zero, then that means both min-entropy leakage and Shannon leakage are greater than zero, hence we can use the lower bound for min-entropy leakage to *detect* side channels. Thus, even if the bound is not tight, it can be very useful in practice.

8.3. Comparison with "Leakwatch"

We also compared our symbolic sampling with a classical (non-symbolic) Monte-Carlo sampling approach as implemented by "Leakwatch" [9]. We made the comparison using `CheckSecret`; the results are shown in Table 2. The size of `t` is fixed at 30, and the "guess" input varies from 10 to 9990. The space of the secret, $\mathcal{D}$, is $\{0 \ldots 10000\}$.

	`CheckSecret`			`CRIME`		noisy `CRIME`		`LawDB`		`Collab`
Size	100	150	200	3	4	3	4	200	300	
$\#PC$	202	302	402	77	799	154	1598	615	1030	2510
Shannon	0.46757	0.46757	0.46757	0.84290	0.92850	0.43598	0.49412	0.77320	1.14203	0.99712
Time	99	364	943	4	36	5	49	106	404	472
Min-Ent.	0.99930	0.99930	0.99930	0.92850	1.00000	0.76553	0.76553	3.32193	3.32193	1.0000
Time	107	412	920	4	37	5	49	158	388	515

TABLE 1: Computation of Shannon and min-entropy leakages using symbolic execution and model counting by Barvinok, for our four example case studies. Leakages are measured in bits, times in seconds.

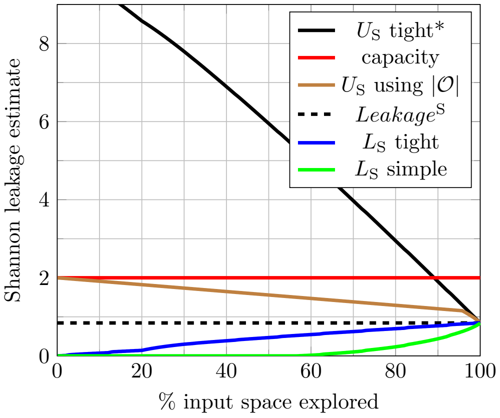

Figure 3: Shannon leakage estimates for `CRIME` (size 3).

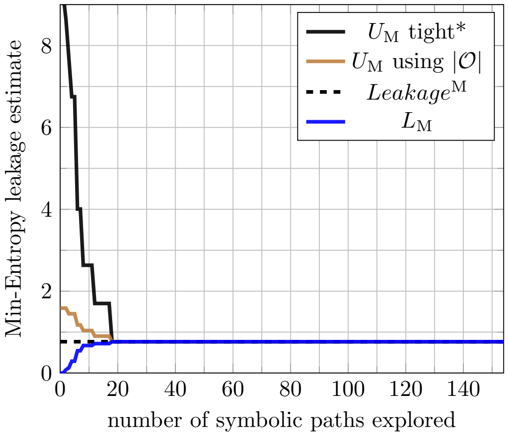

Figure 4: Min-entropy leakage estimates for `CRIME`(size 3).

“Leakwatch” runs using its default settings. Notice first that, for this program, the leakage reported by the symbolic sampling is the real one (the approach coverges quickly); hence we observe that “Leakwatch” wrt symbolic sampling can be very imprecise. In particular in four out of thirteen cases “Leakwatch” wrongly report an absence of leakage: these false negatives can have serious security consequences, and could allow an attacker to go unnoticed and, with repeated runs, obtain the whole password. The second observation is the huge difference in analysis time. The symbolic approach takes on average 6 seconds while “Leakwatch” takes on average over 15 minutes.

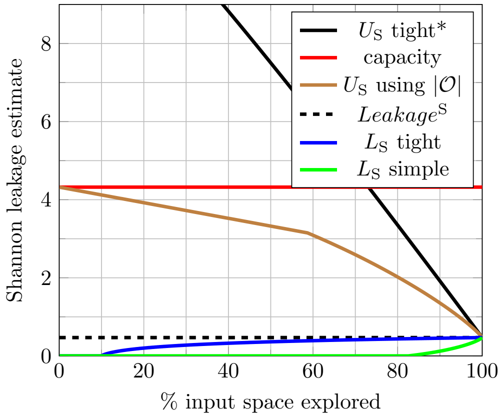

Figure 5: `CheckSecret`’s Shannon leakage estimates (guess=1000, $\mathcal{D} = \{0, \ldots, 10000\}$, size of t= 10).

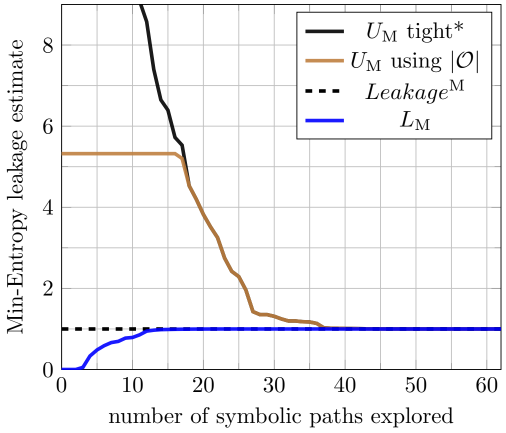

Figure 6: Estimates of `CheckSecret`’s min-entropy leakage (guess=1000, $\mathcal{D}=\{0, \ldots, 10000\}$, size of t=30).

		Guess Values												
		10	100	300	500	1000	3000	5000	8000	9000	9800	9900	9980	9990
LW	Leak	0	0	0.1098	0.2008	0.3840	0.7954	0.9155	0.6410	0.3858	0.0633	0.0044	0	0
	Time	$16m29$	$21m9$	$21m22$	$22m1$	$11m22$	$11m8$	$11m54$	$15m1$	$11m35$	$13m27$	$14m46$	$20m23$	13m16
SPF	Leak	0.0114	0.0806	0.1939	0.2856	0.4676	0.8776	0.9944	0.7145	0.4617	0.1358	0.0761	0.0181	0.0094
	Time	6.465	6.524	6.437	6.544	6.384	5.992	6.849	6.956	7.204	7.244	6.955	6.496	7.15

TABLE 2: Comparison of Shannon leakage evaluation for the `CheckSecret` example (Figure 2) using "Leakwatch" (LW) and symbolic sampling in SPF. Notice time for Leakwatch is in minutes

.

9. Related Work

Symbolic analysis of side channels has been studied in [36], [37]. While these works share a similar symbolic execution platform with our approach, they do not address probabilistic programs or symbolic sampling.

Another related approach [23] uses symbolic analysis and Barvinok model counting to quantify information leakage. That approach is based on a deductive verification system (KeY), and is not fully automated, as it requires user annotations for the KeY proofs. The work applies only to deterministic programs; it translates source code into a logical formula over which projection and model counting are applied; it is not clear how it could be applied to probabilistic programs. They also do not study any sampling.

There have been recent works quantifying leakage using Monte Carlo sampling, e.g. "Leakwatch" [9] and hybrid approaches where sampling is only performed under some conditions, e.g. [19]. However these approaches are based on standard sampling techniques, they are not symbolic and hence of a different nature to ours. Also our symbolic framework allows us to prove stronger theoretical guarantees compared to previous sampling techniques.

There is a significant literature on side-channel analysis, for example [3], [7], [8], [13], [24], [25], [31]. Recent works successfully use abstract interpretation for cache side-channels analysis [14], [26], [30]. Our approach differs from these works by using symbolic execution, symbolic sampling and considering probabilistic behavior.

Probabilistic programs are studied in [28], [32] in the context of enforcement [32] and synthesis [28] of privacy policies, where quantification of leakage becomes relevant. However compared to those works, our focus is on efficient quantification of leakage of side channels: it is not clear how those works advance on this problem. The case studies therein are quite small and even the greedy algorithm SynGrd in [28] relies on computing all observables, which can be prohibitively costly in practice.

Our lower bounds for min-entropy leakage are similar to the iterative approach in [4]. However that paper abstracts information systems into channel matrices whereas we develop a concrete program analysis. Moreover, we consider both min-entropy and Shannon leakages.

Symbolic execution and sampling are also used in [27] with proved bounds for Shannon and min-entropy leakages. However that work considers standard sampling and applies only to deterministic systems; our bounds are for noisy systems and symbolic sampling. Noise introduces some significant technical challenges; also results on standard sampling do not hold for symbolic sampling, hence these works complement each other. Another work [33] provides bounds on information leakage from a sampling analysis, but it only applies to deterministic systems.

Our symbolic sampling is similar to previous work on probabilistic symbolic execution [17] which was done in the context of software safety and reliability. In this paper we provide novel results that allow us to use the information collected with symbolic sampling to compute theoretical lower- and upper- bounds on information leakage.

Conclusion

We presented a symbolic technique for computing leakage in noisy side-channels. We also presented a sampling approach to scale up the technique, and provided upper and lower bounds to the leakage. Future work involves investigating model counting procedures for non-linear constraints [5], [6] which will allow us to expand the applicability of the presented techniques to e.g. cryptographic protocols. We also plan to explore distributed versions of our techniques to further increase scalability.

Acknowledgment. We would like to thank the anonymous reviewers for their valuable comments. This material is based on research sponsored by DARPA under agreement number FA8750-15-2-0087. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] Barvinok library. http://garage.kotnet.org/~skimo/barvinok/.

[2] Giovanni Agosta, Luca Breveglieri, Gerardo Pelosi, and Israel Koren. Countermeasures against branch target buffer attacks, 2007.

[3] Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi. Multi-channel Attacks. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2003.

[4] Miguel E Andrés, Catuscia Palamidessi, Peter Van Rossum, and Geoffrey Smith. Computing the leakage of information-hiding systems. In *TACAS*, pages 373–389. Springer, 2010.

[5] Mateus Borges, Antonio Filieri, Marcelo d'Amorim, Corina S. Păsăreanu, and Willem Visser. Compositional Solution Space Quantification for Probabilistic Software Analysis. In *Proc. of 35th ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '14, pages 123–132, New York, NY, USA, 2014. ACM.

[6] Mateus Borges, Quoc-Sang Phan, Antonio Filieri, and Corina S. Păsăreanu. Model-counting approaches for nonlinear numerical constraints. In *NASA Formal Methods: 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*, pages 131–138. Springer International Publishing, 2017.

[7] David Brumley and Dan Boneh. Remote Timing Attacks Are Practical. In *Proc. of the 12th Conf. on USENIX Security Symposium - Volume 12*, SSYM'03, Berkeley, CA, USA, 2003. USENIX Association.

[8] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In *Proc. of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 191–206, Washington, DC, USA, 2010. IEEE Computer Society.

[9] Tom Chothia, Yusuke Kawamoto, and Chris Novakovic. LeakWatch: Estimating Information Leakage from Java Programs. In *19th European Symposium on Research in Computer Security - Volume 8713*, ESORICS 2014, pages 219–236, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[10] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[11] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Introduction*. Springer, 2008.

[12] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proc. of the 14th international Conf. on Tools and algorithms for the construction and analysis of systems*, TACAS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] Quoc Huy Do, Richard Bubel, and Reiner Hähnle. Exploit Generation for Information Flow Leaks in Object-Oriented Programs. In *ICT Systems Security and Privacy Protection: 30th IFIP TC 11 Intl. Conf., SEC 2015, Hamburg, Germany*, pages 401–415. Springer, 2015.

[14] Goran Doychev, Dominik Feld, Boris Köpf, Laurent Mauborgne, and Jan Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. In *Proc. of 22nd USENIX Conf. on Security*, SEC'13, pages 431–446, Berkeley, CA, USA, 2013. USENIX Association.

[15] Thai Duong and Juliano Rizzo. The CRIME attack. In *Presentation at ekoparty Security Conf.*, 2012.

[16] Antonio Filieri, Corina S. Păsăreanu, and Willem Visser. Reliability analysis in symbolic pathfinder. In *Proc. of the 2013 International Conf. on Software Engineering*, ICSE '13, pages 622–631, Piscataway, NJ, USA, 2013. IEEE Press.

[17] Antonio Filieri, Corina S. Păsăreanu, Willem Visser, and Jaco Geldenhuys. Statistical Symbolic Execution with Informed Sampling. In *Proc. of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 437–448, New York, NY, USA, 2014. ACM.

[18] Jaco Geldenhuys, Matthew B. Dwyer, and Willem Visser. Probabilistic Symbolic Execution. In *Proc. of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, pages 166–176, New York, NY, USA, 2012. ACM.

[19] Yusuke Kawamoto, Fabrizio Biondi, and Axel Legay. Hybrid Statistical Estimation of Mutual Information for Quantifying Information Flow. In *FM*, volume 9995 of *Lecture Notes in Computer Science*, pages 406–425, 2016.

[20] John Kelsey. Compression and Information Leakage of Plaintext. In *Revised Papers from the 9th International Workshop on Fast Software Encryption*, FSE '02, pages 263–276, London, UK, UK, 2002. Springer-Verlag.

[21] MHR Khouzani and Pasquale Malacaria. Leakage-Minimal Design: Universality, Limitations, and Applications. In *30th IEEE Computer Security Foundations Symposium (CSF'17)*, 2017.

[22] James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, July 1976.

[23] Vladimir Klebanov. Precise quantitative information flow analysis–a symbolic approach. *Theoretical Computer Science*, 538:124–139, 2014.

[24] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proc. of the 16th Annual International Cryptology Conf. on Advances in Cryptology*, CRYPTO '96, pages 104–113, London, UK, UK, 1996. Springer-Verlag.

[25] Boris Köpf and David Basin. An Information-theoretic Model for Adaptive Side-channel Attacks. In *Proc. of the 14th ACM Conf. on Computer and Communications Security*, CCS '07, pages 286–296, New York, NY, USA, 2007. ACM.

[26] Boris Köpf, Laurent Mauborgne, and Martín Ochoa. Automatic quantification of cache side-channels. In *Proc. of the 24th international Conf. on Computer Aided Verification*, CAV'12, pages 564–580, Berlin, Heidelberg, 2012. Springer-Verlag.

[27] Boris Köpf and Andrey Rybalchenko. Approximation and Randomization for Quantitative Information-Flow Analysis. In *Proc. of the 23rd IEEE Computer Security Foundations Symposium*, CSF'10, pages 3–14, Washington, DC, USA, 2010. IEEE Computer Society.

[28] Martin Kučera, Petar Tsankov, Timon Gehr, Marco Guarnieri, and Martin Vechev. Synthesis of probabilistic privacy enforcement, 2017.

[29] Pasquale Malacaria. Algebraic foundations for quantitative information flow. *Mathematical Structures in Computer Science*, 25:404–428, February 2015.

[30] Heiko Mantel, Alexandra Weber, and Boris Köpf. A Systematic Study of Cache Side Channels Across AES Implementations, 2017.

[31] Piotr Mardziel, Mário S. Alvim, Michael W. Hicks, and Michael R. Clarkson. Quantifying Information Flow for Dynamic Secrets, 2014.

[32] Piotr Mardziel, Stephen Magill, Michael Hicks, and Mudhakar Srivatsa. Dynamic enforcement of knowledge-based security policies using probabilistic abstract interpretation. *Journal of Computer Security*, 21(4):463–532, 2013.

[33] Stephen McCamant and Michael D. Ernst. Quantitative information flow as network flow capacity, 2008.

[34] David Molnar, Matt Piotrowski, David Schultz, and David Wagner. The program counter security model: Automatic detection and removal of control-flow side channel attacks, 2005.

[35] S Alvim M'rio, Kostas Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions, 2012.

[36] Quoc-Sang Phan, Lucas Bang, Corina S. Păsăreanu, Pasquale Malacaria, and Tevfik Bultan. Synthesis of Adaptive Side-Channel Attacks, August 2017.

[37] Corina S. Păsăreanu, Quoc-Sang Phan, and Pasquale Malacaria. Multi-run Side-Channel Analysis Using Symbolic Execution and Max-SMT, June 2016.

[38] Corina S. Păsăreanu, Willem Visser, David Bushnell, Jaco Geldenhuys, Peter Mehlitz, and Neha Rungta. Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis. *Automated Software Engineering*, pages 1–35, 2013.

[39] Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: Closing Digital Side-channels Through Obfuscated Execution, 2015.

[40] Geoffrey Smith. On the Foundations of Quantitative Information Flow, 2009.