Quantifying Information Flow for Dynamic Secrets

Piotr Mardziel,[†] Mário S. Alvim,[‡] Michael Hicks,[†] and Michael R. Clarkson[*]
[†]University of Maryland, College Park
[‡]Universidade Federal de Minas Gerais
[*]George Washington University

***Abstract*—A metric is proposed for quantifying leakage of information about secrets and about how secrets change over time. The metric is used with a model of information flow for probabilistic, interactive systems with adaptive adversaries. The model and metric are implemented in a probabilistic programming language and used to analyze the security of a family of scenarios, quantifying the impact of various forms of adversarial adaptivity.**

I. Introduction

Quantitative information-flow models [1]–[5] and analyses [6]–[9] typically assume that secret information is *static*. But real-world secrets evolve over time. Passwords, for example, should be be changed periodically. Cryptographic keys have periods after which they must be retired. Memory offsets in address space randomization techniques are periodically regenerated. Medical diagnoses evolve, military convoys move, and mobile phones travel with their owners. Leaking the current value of these secrets is undesirable. But if information leaks about how these secrets change, adversaries might also be able to predict future secrets or infer past secrets. For example, an adversary who learns how people choose their passwords might have an advantage in guessing future passwords. Similarly, an adversary who learns a trajectory can infer future locations. So it is not just the current value of a secret that matters, but also how the secret changes. Methods for quantifying leakage and protecting secrets should, therefore, account for these *dynamics*.

This work initiates the study of quantitative information flow (henceforth, QIF) for dynamic secrets. First, we present a core model of programs that compute with time-varying secrets. We model programs as *probabilistic automata* [10]. These automata are interactive: they accept inputs and produce outputs throughout execution, and the output they produce is a random function of the inputs. To capture the dynamics of secrets, we use *strategy functions* [11] to generate new inputs based on the history of inputs and outputs. For example, a strategy function might yield the GPS coordinates of a high-security user as a function of time, and of the path the user has taken so far.[1]

We then introduce *wait-adaptive* adversaries. These adversaries observe execution of a system, waiting until a point in time at which it appears profitable to attack. For example, an attacker might delay attacking until collecting enough observations of a GPS location to reach a high confidence level about that location. Or an attacker might passively observe application outputs to determine memory layout, and once determined, inject shell code that accesses some secret.

Second, we propose an information-theoretic metric for quantifying flow of dynamic secrets. Our metric can be used to quantify leakage of the current value of the secret, a secret at a particular point in time, the history of secrets, or even the strategy function that produces the secrets. We show how to construct an optimal wait-adaptive adversary with respect to the metric, and how to compute that adversary's expected *gain*, as determined by a scenario-specific *gain function* [14]. These functions consider when, as a result of an attack, the adversary might learn all, some, or no information about dynamic secrets. We show that our metric generalizes previous metrics for quantifying leakage of static secrets, including *vulnerability* [4], *guessing entropy* [15], and *g-vulnerability* [14]. We also show how to limit the power of the adversary, such that it cannot influence inputs, delay attacks, or remember too far into the past.

Finally, we put our model and metric to use by implementing them in a probabilistic programming language and conducting a series of experiments. Several conclusions can be drawn from these experiments:

- Frequent change of a secret can increase leakage, even though intuition might initially suggest that frequent changes should decrease it. The increase occurs when there is an underlying order that can be inferred and used to guess future (or past) secrets.

- *Wait-adaptive* adversaries can learn significantly more information than adversaries who cannot adaptively choose when to attack. So ignoring the adversary's adaptivity (as in prior work on static secrets) might lead one to conclude secrets are safe when they really are not.

- Additionally, the ability of an adversary to delay means that their expected success in attacking a system increases monotonically with time. This is not necessarily true if they do not have the ability to wait.

- *Low-adaptive* adversaries (those who can influence their observations) can learn significantly more information—even exponentially more—than adversaries who cannot.

We proceed as follows. Section II reviews QIF for static secrets and motivates the improvements we propose. Section III presents our model of dynamic secrets, and Section IV presents our metric for leakage. Section V describes our implementation

[1]Our probabilistic model of interaction is a refinement of the nondeterministic model of Clark and Hunt [12], and it is a generalization of the interaction model of O'Neill et al. [13]. See Section VII for details.

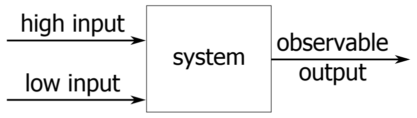
and Section VI presents our experimental results. Section VII discusses related work, and Section VIII concludes.

II. Quantitative Information Flow

Consider a password checker, which grants or forbids access to a user based on whether the password supplied by the user matches the password stored by the system. The password checker must leak secret information, because it must reveal whether the supplied password is correct. Quantifying the amount of information leaked is useful for understanding the security of the password checker—and of other systems that, by design or by technological constraints, must leak information.

A. QIF for static secrets

The classic model for QIF, pioneered by Denning [16], represents a system as an information-theoretic *channel*:



A channel is a probabilistic function. In this model, the system is a channel, because it probabilistically maps a high security (i.e., secret) input and a low security (i.e., public) input to an observable (i.e., public) output. (High security outputs can also be modeled, but we have no need for them in this work.) The adversary is assumed to know this probabilistic function.

The adversary has some initial uncertainty about the high input. By providing low input and observing output, the adversary derives some revised uncertainty about the high input. The change in the adversary's uncertainty is the amount of leakage:

$$\textit{leakage} = \textit{initial uncertainty} - \textit{revised uncertainty}.$$

Uncertainty is typically represented with probability distributions [17]. Specific metrics for QIF use these distributions to calculate a numeric quantity of leakage [1]–[5].

More formally, let $X_{\mathtt{H}}$, $X_{\mathtt{L}}$ and $X_{\mathtt{O}}$ be random variables representing the distribution of high inputs, low inputs, and observables. Given a function $F(X)$ of the uncertainty of X, leakage is calculated as follows:

$$F(X_{\mathtt{H}} \mid X_{\mathtt{L}} = \ell) - F(X_{\mathtt{H}} \mid X_{\mathtt{L}} = \ell, X_{\mathtt{O}}), \quad (1)$$

where ℓ is the low input chosen by the adversary, $F(X_{\mathtt{H}} \mid X_{\mathtt{L}} = \ell)$ is the adversary's initial uncertainty about the high input, and $F(X_{\mathtt{H}} \mid X_{\mathtt{L}} = \ell, X_{\mathtt{O}})$ is the revised uncertainty. As is standard, $F(X_{\mathtt{H}} \mid X_{\mathtt{L}} = \ell, X_{\mathtt{O}})$ is defined to be $\mathbb{E}_{o \leftarrow X_{\mathtt{O}}} \left[F(X_{\mathtt{H}} \mid X_{\mathtt{O}} = o, X_{\mathtt{L}} = \ell) \right]$, where $\mathbb{E}_{x \leftarrow X} [f(x)]$ denotes $\sum_x \Pr(X = x) \cdot f(x)$.

Various instantiations of F have been proposed, including Shannon entropy [1]–[3], [18]–[21], guessing entropy [22], [23], marginal guesswork [24], min-vulnerability [4], [25], and g-leakage [14].

B. Toward QIF for dynamic secrets

There are several ways in which the classic model for QIF is insufficient for reasoning about dynamic secrets:

• **Interactivity:** Since secrets can change, the adversary should be able to choose inputs based on past observations. That is, we want to allow *feedback* from outputs to inputs. The classic model doesn't permit feedback. There are some QIF models for interactivity; we discuss them in Section VII.

• **Input vs. attack:** Classic QIF metrics quantify leakage with respect to a single low input from the adversary. Each input is an *attack* made by the adversary to learn information. But with an interactive system, some low inputs might not be attacks. For example, an adversary might navigate through a website before uploading a maliciously crafted string to launch a SQL injection attack; the navigation inputs themselves are not attacks. Our model naturally supports quantification of leakage at the times when attacks occur.

• **Delayed attack:** Combining the above two features, adversaries should be permitted to adaptively choose when to attack based on their interaction with the system, and this decision process should be considered when quantifying leakage.

• **Moving target:** New secrets potentially replace old secrets. The classic model cannot handle these "moving targets." To quantify leakage about moving-target secrets, we need a model of how secrets evolve over time. Prior work [26], [27] has considered leakage only about the entire stream of secrets, rather than a particular value of a secret at a particular time.

To address these insufficiencies in the classic model, we introduce a new model for QIF of dynamic secrets.

III. Model of Dynamic Secrets

Our model of dynamic secrets involves a *system*, which executes within a *context*. The system represents a program, such as a password checker. The context represents the interaction of the program with its environment, which includes the users who provide inputs.

A. Systems as probabilistic automata

As in the classic model of QIF, a system accepts a high input and a low input, and produces an observable output. Let $\mathcal{H}$, $\mathcal{L}$, and $\mathcal{O}$ be finite sets of high inputs, low inputs, and observable outputs.

Our model adds a notion of logical time to the classic model. At each time step t, three *events* occur sequentially: (i) the system accepts a high input h_t; (ii) the system accepts a low input ℓ_t; and (iii) the system produces an observable output o_t. An execution lasting T time steps thus produces an execution *history* (synonomously, a *trace*) of the following form:

$$\underbrace{h_1\ \ell_1\ o_1}_{t=1}\ \underbrace{h_2\ \ell_2\ o_2}_{t=2}\ \cdots\ \underbrace{h_T\ \ell_T\ o_T}_{t=T}.$$

Our assumption of cyclic alternation between high inputs, low inputs and observable outputs does not preclude systems where these events happen in other orders. To model such systems,

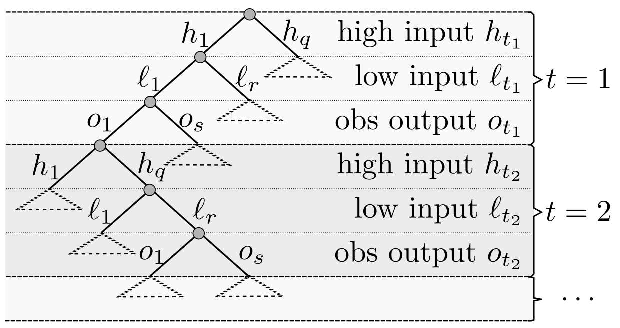


Fig. 1. An execution of probabilistic automaton corresponding to a system. Each edge is also labeled with a probability, which is omitted from the figure for simplicity.

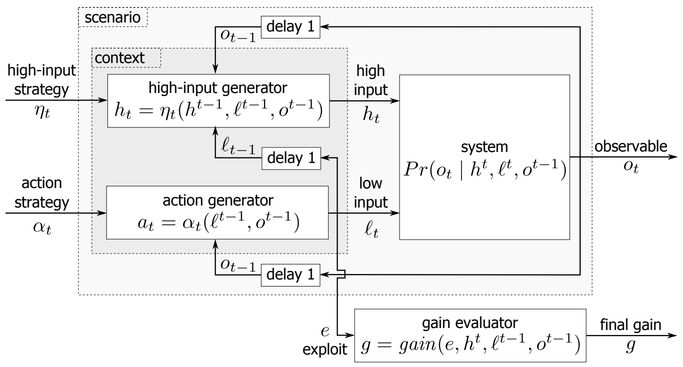

scenario
context
high-input strategy η_t
high-input generator $h_t = \eta_t(h^{t-1}, \ell^{t-1}, o^{t-1})$
delay 1
o_{t-1}
ℓ_{t-1}
high input h_t
system $Pr(o_t \mid h^t, \ell^t, o^{t-1})$
observable o_t
action strategy α_t
action generator $a_t = \alpha_t(\ell^{t-1}, o^{t-1})$
low input ℓ_t
e exploit
gain evaluator $g = gain(e, h^t, \ell^{t-1}, o^{t-1})$
final gain g

Fig. 2. Model of dynamic secrets. The arrows feeding high inputs, low inputs and observables to the gain evaluator are omitted for simplicity.

it suffices to define dummy inputs or outputs that are used whenever an execution skips some event.

A system can be represented using a *probabilistic automaton* [10].[2] The execution of such an automaton can be represented as a tree where any path from the root to a leaf corresponds to an execution history, as depicted in Figure 1. Each edge in the tree corresponds to an event in the history. Let x^t denote the sequence of events of type x up to time t—for example, $h^t = h_1, \ldots, h_t$. Denote the probability of an event as follows:

- **High inputs:** $\Pr(h_t \mid h^{t-1}, \ell^{t-1}, o^{t-1})$ is the probability of high input h_t occurring, conditioned on the execution history $(h^{t-1}, \ell^{t-1}, o^{t-1})$ through time $t-1$.

- **Low inputs:** $\Pr(\ell_t \mid \ell^{t-1}, o^{t-1})$ is the probability of low input ℓ_t occurring, conditioned on the public execution history (ℓ^{t-1}, o^{t-1}) through time $t-1$. Since this probability is not conditioned on h^{t-1}, low inputs cannot depend on past high inputs.

- **Observable outputs:** $\Pr(o_t \mid h^t, \ell^t, o^{t-1})$ is the probability of the system producing observable output o_t, conditioned on the execution history $(h^{t-1}, \ell^{t-1}, o^{t-1})$ up to time $t-1$, as well as the high input h_t and low input ℓ_t occurring at time t.

B. Interaction of a system with the environment

[2]In Segala's [10] nomenclature, the type of probabilistic automaton we use in this paper is *fully probabilistic*.

A system receives inputs from its environment, and it yields outputs back to that environment. The environment includes agents, such as the adversary. With the password checker, for example, a password file might be a source of high inputs, and the adversary might be a source of low inputs—specifically, of guesses in an online guessing attack. Figure 2 depicts the interaction of a system with its environment. We now explain each part of the figure.

Define an execution *context* to be a pair of functions—called the *high-input generator* and the *action generator*—that generate inputs for the system. These functions represent the high and low agents in the environment. The high-input generator produces a high input h_t at each time step t using a *high-input strategy* [11], which is a function η_t of the history $(h^{t-1}, \ell^{t-1}, o^{t-1})$ of execution. Note that a different high-input strategy (and action strategy, described below) may be used at each time step.

The action generator models the distinction between attacks and inputs. At each time step, the action generator produces a new input on behalf of the adversary, resulting in a new observable from the system. That observable is fed back into the action generator at the next time step. Eventually, the adversary has gathered enough observations and commits to an attack, represented by the generator producing an *exploit*. (We leave modeling interleaved attacks and low inputs as future work.) Let $\mathcal{E}$ be set of exploits. Define an *action* to be either a low input or an exploit, and let $\mathcal{A}$ be the set of actions, where $\mathcal{A} = \mathcal{L} \cup \mathcal{E}$. At each time step, the action generator produces an action a using an *action strategy*, which is a function α_t of the public history (ℓ^{t-1}, o^{t-1}) of execution.

The adversary in our model is *wait adaptive*: it can pick the best time to attack. An adversary that is not wait adaptive, as in the classic model of QIF, can attack only at a fixed time. Similarly, the adversary in our model is *low adaptive*: it can generate low inputs that influence the system. An adversary that is not low adaptive would have to passively observe the system prior to attack.

Our model makes manifest the two input generators that make up the context, whereas these would be implicit in a probabilistic automaton. Making them explicit clearly identifies the agents we are interested in, along with their capabilities (i.e., what they are able to observe). Just as the adversary can learn about particular high-input values through observations, he can learn about the high-input strategy that generated them, giving him more predictive power. We discuss quantifying information about the strategy in Section IV-E and the experiment in Section VI-F shows how learning the strategy is important for learning the (past, present, and future) secrets.

Our channel model assumes that strategies are deterministic functions. Nonetheless, we can emulate probabilistic strategies by introducing a probability distribution on deterministic strategies. The equivalence between the two approaches is stated by Corollary 1 in Section III-E. Deterministic strategies are more convenient for proving mathematical properties of the model, so we use them in the remainder of this section. Probabilistic strategies, on the other hand, are more convenient for expressing our metrics, and also make it more straightforward to reason about increasing knowledge of the strategy. Section IV and following sections use probabilistic strategies.

C. Success of the adversary

Once the adversary commits to an exploit e, no more observations take place. The success of the adversary is now determined. Define a *scenario* to be the history $h^{t-1}, \ell^{t-1}, o^{t-1}$ of execution up to the exploit. The *gain evaluator*, shown in Figure 2, is a function *gain* of exploit e and scenario h^t, ℓ^{t-1}, o^{t-1} (recall that when the action $a_t = e$ is produced the high input h_t is already available, but not the low input ℓ_t or the observable o_t). It yields a real number g representing the success of the exploit.[3] Some prior metrics for QIF consider an exploit to be successful iff the adversary perfectly guesses the secret. But more sophisticated gain functions can quantify the success of the adversary in guessing part of a secret, guessing a secret approximately, or guessing a past secret [14].

D. Example: Password checker

We now formally capture a password checker in our model. Another detailed example, on *stakeouts and raids*, will be used in our experimental results in Section VI.

Let us model a password checker as a system receiving as high input the real password and as low input a guess provided by a user (possibly the adversary). The system can produce one of two observables: *acceptance*, in case the guess matches the password, and *rejection* otherwise. Formally: $\mathcal{H}$ is the set of possible passwords, $\mathcal{L}$ is the set of possible guesses (presumably the same set as the set of passwords), and $\mathcal{O} = \{\texttt{accept}, \texttt{reject}\}$. The system channel $\{\Pr(o_t \mid h^t, \ell^t, o^{t-1})\}_{t=1}^{T}$ is such that at each time step t, $\Pr(\texttt{accept}) = 1$ iff $h_t = \ell_t$, and $\Pr(\texttt{reject}) = 1$ iff $h_t \neq \ell_t$.

The system manager acts as the high-input generator. If he keeps a log of attempted logins into the system, including failed ones, he can compile a list of the most common unauthorized guesses for a password (e.g., birth dates, telephone numbers, dictionary words). Assume that the manager requires the password to be changed regularly, following two rules: (i) a new password cannot coincide with any of the 5 previous passwords; and (ii) a password cannot coincide with any of the 10 most common guesses previously attempted by the adversary. The password changing policy can be captured in our model as a high-input strategy depending on the past history of high inputs (what passwords have already been used), on the past history of low inputs (what guesses are most common), and on the past history of observables (failed guesses are included in the log). Formally, a password h_t is generated according to a strategy $\eta_t(h^{t-1}, \ell^{t-1}, o^{t-1})$ in such a way that $h_t \neq h_{t-5}, \cdots, h_{t-1}$, and if h_t does not coincide with the 10 most frequent $\ell \in \ell^{t-1}$.

As for the adversary, he naturally produces new guesses based on whether past attempts were successful. The action strategy depends on the past history of low inputs, and on the past history of observables. For instance, because passwords are generally not refreshed every single time a log in is attempted (but rather every once in a while), it may be reasonable to avoid a failed guess for some period before trying it again. If a failed guess cannot be re-used for (say) 5 time units, guesses can be formally generated by some strategy $\ell_t = \alpha_t(\ell^{t-1}, o^{t-1})$ where $\ell_t \neq \ell_{t-5}, \cdots, \ell_{t-1}$.

Once the adversary has gathered enough information by feeding low inputs to the system and observing the produced outputs, he can commit to an attack by picking an exploit from a set $\mathcal{E}$. The set of exploits for a password checker could coincide with the set $\mathcal{L}$ of low inputs, since the most usual attack is to try to log in into the system using the password. Our model, however, allows the reasoning not only about how much information about current passwords leak, but also about how much information about the high-input strategy leaks, and this knowledge can be exploited by the adversary. In the next section we will discuss the mathematical foundations of how this is possible, and concrete experiments are discussed in Section VI.

E. On the mathematical soundness of the model

The general behavior of the system after T time steps can be completely described by the joint distribution $\Pr(h^T, \ell^T, o^T)$ derived from the execution of the probabilistic automaton representing the system:

$$\begin{aligned}\Pr(h^T, \ell^T, o^T) = \prod_{t=1}^{T} [\ &\Pr(h_t \mid h^{t-1}, \ell^{t-1}, o^{t-1}) \\ &\cdot \Pr(\ell_t \mid \ell^{t-1}, o^{t-1}) \\ &\cdot \Pr(\ell_t \mid h^t, \ell^t, o^{t-1})\ ]\end{aligned} \tag{2}$$

The distribution p, however, is defined over the history of high inputs, low inputs and observables only, and does not explicitly model the way the context generates high inputs and low inputs using strategies. Since we would also like to reason about information flows from these, we employ a technique proposed by Tatikonda and Mitter [28], and applied by Alvim et al. to interactive systems [27], to extend (2) to also include random variables η_t and α_t corresponding to the choice of a high-input strategy and of an action strategy, respectively, at each time t. The details are given in the Appendix. The upshot is that our model (Figure 2) is well suited to information theoretically modeling the systems we are interested in,[4] which is attested to by the following key results.

Theorem 1 shows that it is possible to bring together the random variables h^T, ℓ^T, o^T describing the behavior of the system and the random variables η^T, α^T describing the behavior of its context into a joint probability distribution $Q(\eta^T, \alpha^T, h^T, \ell^T, o^T)$. The existence of Q allows us to condition the occurrence of system-related events on the occurrence of context-related events, and vice-versa. For instance, we can

[3]The gain evaluator does not have access to future secrets values h_u, where $u > t$. So even if the adversary determines at time t what the high input will be at time $u > t$, the adversary must wait until time u to realize the gain. To give a real-world example, a thief who learns today that a house will be unoccupied next Tuesday still has to wait until next Tuesday to rob the house.

[4] This technique can also be used to facilitate the computation of the maximum leakage of the system over all possible probability distributions on secrets. As shown by Alvim et al. [27], when there is *feedback* from the system to its context, (2) does not specify a channel that is invariant with respect to the distribution on low and high inputs, which is required in traditional information-theoretic approaches for calculating maximum leakage. Tatikonda and Mitter also solve this problem; a discussion on the matter is beyond the scope of this paper.

use Q to reason about how much information the observables produced by the system reveal about the high-input strategies of its context.

Theorem 1. *Given a system and a context, there is a unique joint probability distribution $Q(\eta^T, \alpha^T, h^T, \ell^T, o^T)$ capturing the interactions of all elements in the model of Figure 2.*

Theorem 2 shows how to construct, from the probabilistic automaton describing a system, a context that explicitly captures how high and low inputs are probabilistically generated by agents external to the system.

Theorem 2. *Given the probabilistic automaton for a system, it is possible to construct a context that captures the probabilistic generation of high and low inputs fed to the system. This context induces a joint probability distribution $Q(\eta^T, \alpha^T, h^T, \ell^T, o^T)$ consistent with the model of Figure 2.*

The proof of Theorem 2 constructs the appropriate context as a set of deterministic high-input strategies and a set of action strategies, each endowed with an appropriately defined probability distribution. An immediate consequence of this construction is stated in the following corollary.

Corollary 1. *Probabilistic strategies for generating high and low inputs can be captured by appropriately defined probability distributions on deterministic strategies.*

IV. Quantifying security

Having defined the full model we can now derive from it means of quantifying security. To facilitate this, we represent our model as a *probabilistic program*, which precisely describes the joint distribution it induces. Using this notation makes it easier to define particular scenarios precisely, as is done in Section V, and maps closely what we do in our implementation. In particular, we literally implement this model in a probabilistic programming language and use it to compute metrics of interest.

Given this new presentation of the model, we show how to quantify security in terms of the *gain* adversaries are expected to achieve while interacting with a system. We will describe this expectation in terms of the optimal adversary that strives to achieve the most gain (Section IV-C).[5] In Sections IV-D and IV-E we show this general definition of security expresses and extends the existing metrics of min-vulnerability, g-vulnerability, and guessing entropy.

A. Probabilistic programming

Probabilistic programs permit the expression of probability distributions using familiar programming language notation. In this paper, we will express probabilistic programs in slightly sugared OCaml, an ML-style functional programming language [29]. In essence, probabilistic programs are just normal programs that employ randomness. For example, the following program employs the `random_int` function to draw a random integer from the uniform distribution of non-negative integers. Each run of the `gen_stash` program can thus be viewed as sampling a number from a uniform distribution of integers between 0 and 7:

```
let gen_stash () =
  let real_loc = (random_int () mod 8) in real_loc
```

The `gen_stash` program can be seen as a random variable $X_{\texttt{gen\_stash ()}}$, whose sample space is uniformly distributed between 0 and 7. We write $X_{\texttt{exp}}$ to denote a random variable whose sampling function is the probabilistic program `exp`.

While `gen_stash` takes no arguments, in general functions may take arguments and these arguments might themselves be probabilistic. Consider the following example:

```
let guess (realval_H: int) (guessval_L: int) =
  let correct_L = (realval_H == guessval_L) in
    correct_L
```

The `guess` program takes two arguments and returns whether they are equal. Thus we can define random variable $X_{\texttt{guess (gen\_stash()) 5}}$ over booleans, where **`true`** will be drawn 1/8 of the time, and **`false`** will be drawn otherwise.

The distribution of this random variable obviously depends on the distribution of the random variable corresponding to `gen_stash ()`, so we can condition the probability of an outcome on the latter given an outcome of the former; e.g.,

$$\Pr\left(X_{\texttt{gen\_stash ()}} \mid X_{\texttt{guess (gen\_stash ()) 5}} = \textbf{false}\right)$$

would be the uniform distribution of integers between 0 and 7, but not including 5.

Other notational conventions will be introduced as we go.

[5]Appendix B also considers a *memory-limited* adversary that strives to achieve the most gain.

```
type time = int
type history =
  {t: time; tmax: time;
   highs: ℋ list;
   lows: ℒ list;
   obss: 𝒪 list
   atk: ℰ option}
type 𝒜 = Wait of ℒ | Attack of ℰ

type sysf = history → 𝒪
type highf = history → ℋ
type actf = int → ℒ list → 𝒪 list → 𝒜
type gainf = history → float
```

Fig. 3. Types used by the probabilistic program implementing the model.

B. The model as a probabilistic program

Now we consider how to express the model of Figure 2 as a probabilistic program.

Elements of the model: The types of values in the program are $\mathcal{H}$, $\mathcal{L}$, $\mathcal{O}$, and $\mathcal{E}$, as in the information-theoretic presentation. Figure 3 gives the types of other elements used in the model. Define a record type *`history`* to be the history of execution: The first field of the record contains the current time; the next three contain the high inputs, low inputs, and observations produced thus far; and the last contains the final exploit.[6] At each time step, the adversary will produce an *action* $\mathcal{A}$, which is either a low input or an exploit.

```
1  let scenario (T: time) (system: sysf)
2    (high_func: highf) (strat_func: actf) =
3
4    let hist = {t = 0; tmax = T; atk = None;
5                highs = []; lows = []; obss = []} in
6
7    while hist.t <= T && hist.atk = None do
8      hist.t <- hist.t + 1;
9
10     let new_high = high_func hist in
11
12     hist.highs <- hist.highs @ [new_high];
13
14     let new_action =
15       strat_func hist.t hist.lows hist.obss in
16
17     match new_action with
18     | Attack exp ->
19       hist.atk <- Some exp
20     | Wait new_low ->
21       hist.lows <- hist.lows @ [new_low];
22       let new_obs = system hist in
23         hist.obss <- hist.obss @ [new_obs]
24   done;
25   hist
26
27 let evaluate (T: time)
28          (system: sysf)
29          (high_func: highf)
30          (gain_func: gainf)
31          (strat_func: actf) =
32   let hist = scenario T system
33                  high_func strat_func in
34   let gain = match hist.atk with
35     | Some exp -> gain_func hist exp
36     | None -> −∞ in
37   gain
```

Fig. 4. The model as a probabilistic program.

Operation of the model: Figure 4 presents the model as a probabilistic program, using two functions. The first, `scenario`, corresponds to the identically named element in Figure 2 while the second, `evaluate`, uses `scenario` and then evaluates the resulting gain from the history produced.

The `scenario` function takes four arguments. The first, `T`, is maximum number of time steps to consider. The second is the system being modeled, which is a function of type *sysf* (all types are defined in Figure 3). The last two arguments comprise the *context*, i.e., the high-input generation function (of type *highf*) and the adversarial strategy function (of type *actf*). The scenario starts with the initial history at time 0, with empty lists, and no attack. The loop at line 7 captures the iterations of Figure 2, updating the history for up to T iterations, or until the adversary produces an attack. In each iteration, a new secret is produced (Line 10), an adversary action is computed (Line 14), and if the action is to wait, a new observation is made (Line 22).[7] This function returns the full history when it completes.

The `evaluate` function computes how successful the adversary was by applying their exploit to the gain function (Line 35), which has type *gainf*. Evaluation returns minimal gain if the history does not contain an exploit or otherwise evaluates the gain function.

Comparing to the information theoretic model: Our probabilistic program corresponds quite closely to the information theoretic model of the previous section. The one difference is that the probabilistic program directly employs a single high-input generation function `high_func` that can itself be randomized, as opposed to using a randomized stream of deterministic high-input generation functions (and likewise for the action generator). As per Corollary 1, doing this is still faithful to the model, and it turns out to be more tractable (and convenient) to implement.

C. The general metric

Now we turn our attention to defining a general quantitative metric of information leaked by the model. In what follows we will fix all the `evaluate` parameters except the last two (the gain function and the strategy function). We will use the expression

```
model = evaluate T system high_func
```

as an instantiation of the fixed parameters. The expected gain is thus the expectation of the random variable $X_{\texttt{model gain\_func strat\_func}}$, or $\mathbb{E}\left[X_{\texttt{model gain\_func strat\_func}}\right]$.

An information flow metric is most useful when considered from the point of view of a powerful adversary. In particular, we are most interested in what the system can be expected to leak when interacting with an adversary employing the *optimal* strategy, which we denote as `opt_strat`.

Definition 3. The *dynamic gain* is the gain an optimal adversary is expected to achieve under the parameters of a `model` from a gain function `gain_func`. We will note this quantity as below:

$$\mathbb{D}_{\texttt{gain\_func}}(\texttt{model}) \overset{\text{def}}{=} \max_{\texttt{s}\in \mathit{actf}} \mathbb{E}\left[X_{\texttt{model gain\_func s}}\right] = \mathbb{E}\left[X_{\texttt{model gain\_func opt\_strat}}\right]$$

One way to compute the dynamic gain is to consider essentially all adversary choices in the joint distribution fully describing the model, picking out the best ones. The best choices made thus will both define the optimal adversary `opt_strat` and the gain they expect to achieve. The rest of this subsection considers an algorithm for doing this.

To start, note the adversary's strategy has direct control of three things: when to attack, the attack parameter itself, and low inputs to the system. We can introduce all possible choices via a random strategy that tries everything.[8]

```
let rand_strat: actf =
  fun (t: time) (lows: ℒ list) (obss: 𝒪 list): 𝒜 ->
    if flip 0.5 then
      Attack (uniform_select ℰlist)
```

[6]type α *option* is an OCaml type whose values are either `None`, or `Some x` where `x` is of type α.

[7]The `@` operator appends two lists. We will use OCaml's array indexing notation `a.(i)` for lists as well. That is, `l.(i) = nth l i`. We also define `last l = l.(length l - 1)` to get the last element of a list.

[8]We assume $\mathcal{E}$list and $\mathcal{L}$list are lists of all attacks and all low inputs respectively. The function `flip p` is a random coin flip returning true or false with probabilities p and $1-p$ respectively. `uniform_select` uniformly picks an element from a given list.

```
    else
      Wait (uniform_select ℒlist)
```

Given a `gain_func`, the evaluation of `model gain_func rand_strat` produces the random variable $X_{\texttt{model gain\_func rand\_strat}}$. Along with the final return value of this expression we will also make use of the random variables for some other expressions that are involved in this computation. Namely, we will make use of the joint distribution over the final `gain` variable, and the `atk`, `highs`, `lows`, and `obss` fields of `hist`:

$$\Pr(X_{\texttt{gain}}, X_{\texttt{hist.atk}}, X_{\texttt{hist.highs}}, X_{\texttt{hist.lows}}, X_{\texttt{hist.obss}})$$

To compute the dynamic gain for the optimal adversary, we define a map G (for gain) from lists of low inputs and observations to the expected gain the optimal adversary will receive given a choice of low inputs, and occurrence of observations. Starting from full histories (lists of length `T`) and working backwards to the initial history (empty lists), this map's construction parallels the optimal choices an adversary makes. Though the joint distribution is constructed from completely random choices, these are effectively removed from the optimal quantities via conditioning.

- When `length lows = length obss = T` we fill in the map `G` as follows:

$$\begin{aligned} G[\texttt{lows, obss}] &\stackrel{\text{def}}{=} \\ \max_{e\in\mathcal{E}} \mathbb{E}[X_{\texttt{gain}} \mid\ & X_{\texttt{hist.lows}} = \texttt{lows}, \\ & X_{\texttt{hist.obss}} = \texttt{obss}, \\ & X_{\texttt{hist.atk}} = \texttt{Some e}] \end{aligned} \tag{3}$$

The expression determines the best attack for an adversary that has chosen the list of lows `lows` and observed `obss` as a result. There is no need to consider a choice of waiting at time `T` as that would result in minimum gain.

- Otherwise, if `length lows = length obss = n < T` we define:

$$\begin{aligned} G[\texttt{lows, obss}] \stackrel{\text{def}}{=} \max\{ & \\ & \left.\begin{aligned}\max_{e\in\mathcal{E}} \mathbb{E}[X_{\texttt{gain}} \mid & X_{\texttt{hist.lows}} = \texttt{lows}, \\ & X_{\texttt{hist.obss}} = \texttt{obss}, \\ & X_{\texttt{hist.atk}} = \texttt{Some e}],\end{aligned}\right\}\text{attack now} \\ & \left.\begin{aligned}\max_{\texttt{l}\in\mathcal{L}} & \mathop{\mathbb{E}}_{\texttt{o}\leftarrow O^{n+}_{\texttt{lows,obss}}(l)} \\ & G[\texttt{lows @ [l], obss @ [o]}]\}\end{aligned}\right\}\text{wait} \end{aligned} \tag{4}$$

The outer maximization describes the adaptive choice of either attacking the system at time `n` or waiting. The optimization of attack is identical to that of Equation 3 above. Further, the choice of waiting includes the further adaptive choice of the next low input. The notation $O^{n+}_{\texttt{lows,obss}}(\texttt{l})$ in this optimization is a random variable representing the distribution of observables at the next time step given a particular choice `l` of low input:

$$\begin{aligned} & \Pr\left(O^{n+}_{\texttt{lows,obss}}(\texttt{l}) = \texttt{o}\right) \stackrel{\text{def}}{=} \\ & \Pr(X_{\texttt{hist.obss.(n+1)}} = \texttt{o} \mid \\ & \qquad X_{\texttt{hist.lows}} =_{(n+1)} \texttt{lows @ [l]}, \\ & \qquad X_{\texttt{hist.obss}} =_n \texttt{obss}) \end{aligned} \tag{5}$$

The $=_n$ notation above corresponds to equality of the first n elements of lists: $X_{\texttt{list1}} =_n \texttt{list2} \stackrel{\text{def}}{=} X_{\texttt{take n list1}} = \texttt{take n list2}$.

For non-final actions, the adversary can either attack now, optimizing their action `a`, or wait, optimizing their choice of low input `l` for the next observation. The expectation in the latter optimization is due to the fact that the adversary does not know the secret part of the history nor has control over the potentially non-deterministic aspects of the system function.

Constructing the map `G` backwards in the length of histories (due to the recursion in the definition of G) eventually produces $G[[],[]]$ and this is the expected gain of the optimal attacker in a model, or $\mathbb{D}_{\texttt{gain\_func}}(\texttt{model})$.

D. Expressing existing metrics

Here we show how our metric for optimal adversary gain subsumes existing metrics, in particular, min-vulnerability, g-vulnerability, and guessing entropy. To do this, we define a scenario that corresponds to the classic scenario, a gain function specific to each metric, and prove that the dynamic gain for these matches the standard metric. Further details (and proofs) are given in the Appendix.

The restricted model is defined as follows. First, the high-input generation function is a constant function, since the secret never changes. Second, the system is only permitted to observe the high value and the attack (not past observations or low inputs, which are ignored). Finally, each gain function is defined to provide positive gain only when carried out at time `T`, thus eliminating any benefit of waiting. Under these restrictions, Equations 4 and 5 can be rewritten resulting in a simpler definition of gain. In what follows we omit the `lows` parts, and write `secret` to express `last hist.highs`, the sole unchanging secret value. We will refer to the expressions `model` and gain functions `gain_func` that instantiate parameters in the restricted manner enumerated here as *static*.

Lemma 1. *If `model` and `gain_func` are static then dynamic gain (which would be more appropriately named static gain) simplifies to the following.*

$$\begin{aligned} \mathbb{D}_{\mathit{gain\_func}}(\mathit{model}) = & \\ \mathop{\mathbb{E}}_{\mathit{obss}\leftarrow X_{\mathit{hist.obss}}} \max_{e\in\mathcal{E}} \mathbb{E}(X_{\mathit{gain}} \mid & \\ & X_{\mathit{hist.obss}} = \mathit{obss}, \\ & X_{\mathit{hist.atk}} = \mathit{Some\ e}) \end{aligned}$$

Now we turn to the particular metrics.

Min-vulnerability: The notion of min-vulnerability corresponds to an equality gain function.

```
let gain_min_vul: gainf =
  fun (hist: history) (exp: ℰ) ->
    if secret == exp then 1.0 else 0.0
```

The goal of the attacker assumed in min-vulnerability is evident from `gain_min_vul`; they are directly guessing the secret, and they only have one chance to do it.

Theorem 4. *In a static* `model`*, the min-vulnerability of the secret conditioned on the observations is equivalent to dynamic gain using the* `gain_min_vul` *gain function.*

$$\mathbb{D}_{gain\_min\_vul}(model) = \mathbb{V}(X_{secret} \mid X_{hist.obss})$$

G-vulnerabiity: Generalized gain functions can be used to evaluate metrics in a more fine-grained manner, leading to a metric called g-vulnerability. This metric can also be expressed in terms of the static model. Let `g` be a generalized gain function, returning a `float` between 0.0 and 1.0, then we have:

```
let gain_gen_gain (g: ℋ → ℰ → float): gainf =
  fun (hist: history) (exp: ℰ) : float ->
  g secret exp
```

The difference between expected gain and g-vulnerability are non-existent in the static model. The gain of a system corresponds exactly to g-vulnerability of `g`, written $\mathbb{V}_{g}(\cdot)$.

Theorem 5. *In a static* `model` *the g-vulnerability of* `g` *is equivalent to dynamic gain using* `gain_gen_gain g` *gain function.*

$$\mathbb{D}_{gain\_gen\_gain}(model) = \mathbb{V}_{g}(X_{secret} \mid X_{hist.obss})$$

Guessing-entropy: Guessing entropy, characterizing the expected number of guesses an optimal adversary will need in order to guess the secret, can also be expressed in terms of the static model. We let attacks be lists of highs (really permutations of all highs). The attack permutation corresponds to an order in which secrets are to be guessed. We then define expected gain to be proportional to how early in that list of guesses is.

```
type ℰ = ℋ list

let pos_of (secret: ℋ) (exp: ℋ list) =
  (* compute the position of secret in exp *)

let gain_guess_ent: gainf =
  fun (hist: history) (exp: ℰ) =
    -1.0 * (1.0 + (pos_of secret exp))
```

Note that we negate the gain as an adversary would optimize for the minimum number of guesses, not the maximum. Guessing entropy, written $\mathbb{G}(X)$ is related to dynamic gain as follows.

Theorem 6. *In a static* `model`*, guessing entropy is equivalent to (the negation of) dynamic gain using the* `gain_guess_ent` *gain function.*

$$-\mathbb{D}_{gain\_guess\_ent}(model) = \mathbb{G}(X_{secret} \mid X_{hist.obss})$$

E. Extending existing metrics

Our model lets us take existing information flow metrics and extend their purview along two directions: temporal variations in the target of the attack and the attacker capabilities. Each of these extensions can be specified in terms of the more general scenario and gain functions permitted in our model. As such, we preserve the goal of an existing metric but apply it to situations in which the existing definitions is insufficient.

The first direction gives us several choices as to what about the present, past, or future is the intended attack target. We will briefly cover four categories: *moving target*, *specific past*, *historical*, and *change inference*.

1) **Moving target:** The target of the attack is the current high value. Defining the gain in terms of the most recent high value, rather than the original high value, produces the moving-target equivalents of min-vulnerability, g-vulnerability, and guessing entropy; i.e., we use the same gain functions as Section IV-D but with non-constant high-input generation functions.
2) **Specific past gain:** The target of attack is the high value at some fixed point in time (rather than the most recent one). The gain function would thus evaluate success against this secret, independent of the current secret.
3) **Historical gain:** The target of attack is the entire history of high values up to the present time. An attack on the whole history can be formulated by extending the set of attacks to include lists of all lengths up to `tmax` and specifying dynamic gain in terms of equality of this attack and the history.
4) **Change inference:** The target of the attack is not the high values, but the high-input generation function that produces them. As it stands, an adversary's interactions will increase his knowledge of the high-input generation function, but such knowledge is only indirectly quantified by knowledge of the high values themselves. To quantify knowledge of the high-input generation function directly, we could slightly extend the definition of gain functions:

   ```
   type gainf = history → highf → float
   ```

   Such discrimination, however, would need to be syntactic and not semantic (two functions that are identical in semantics would be considered separate).

On the second axis of extension we have capabilities by which the adversary can interact, or influence, the system prior to attacking. Our model permits consideration of *low adaptive* and *wait adaptive* adversaries, who can carefully choose how to interact with the system prior to attacking it, and/or can wait for the best moment to attack.

Section VI conducts experiments that explore some of these metrics.

V. Implementation

We have implemented our model using a simple monadic embedding of probabilistic computing [30] in OCaml, as per Kiselyov and Shan [31]. The basic approach is as follows. The `model` function, translated into monadic style, is probabilistically evaluated to produce the full joint distribution over the history up to some time T. From this joint distribution the optimal adversary's expected gain is constructed according to the algorithm in Section IV-C. The implementation (and experiments from the next section) are available online.[9]

In more detail, our implementation works as follows. We represent a distribution as either a map from values to floats (a slight extension of `PMap` of the extlib library[10]) or an

[9]https://github.com/plum-umd/qif/tree/master/oakland14

[10]ocaml-extlib: https://code.google.com/p/ocaml-extlib

enumerator of value and float tuples (Enum of the extlib library), converting between the two at various points in the simulation.

```
module M = struct
  type 'a dist = ('a, float) PMap.t
  ...
module E = struct
  type 'a dist = ('a * float) Enum.t
  ...
```

The former is used to represent a full mapping of every possible value to its probability but requires all these values to be stored in memory. The latter has low memory requirements and is used before the probabilities of values need to be retrieved. These distributions are manipulated in a monadic style using functions such as:

```
val bind: 'a dist -> ('a -> 'b dist) -> 'b dist
val return: 'a -> 'a dist
val bind_flip: float -> (bool -> 'b dist) -> 'b dist
val bind_uniform:
  int -> int -> (int -> 'b dist) -> 'b dist
val bind_uniform_select:
  'a list -> ('a -> 'b dist) -> 'b dist
```

The first two are standard. The next creates a random coin flip and continues the computation over this flip to produce a distribution. The next does the same with a random integer in a given range and the last uniformly picks a value from a list to continue the computation with. The program below shows how these functions are used to compute the probability that the sum of two dice is 10.

```
let d_dice1 = M.bind_uniform 1 6 M.return in
let d_dice2 = M.bind_uniform 1 6 M.return in
let d_sum = M.bind d_dice1 (fun dice1 ->
  M.bind d_dice2 (fun dice2 ->
    M.return (dice1 + dice2))) in
let prob_10 = PMap.find d_sum 10
```

In the first two lines, two distributions are created that map integers 1 through 6 to the probability $1/6$. In the third line bind is used to take an initial distribution d_dice1 and a function that will continue the computation of a distribution starting from a value of the first. This continuation gets called once for each possible value in d_dice1 and each time produces another distribution. All 6 of these are merged into one when this bind finishes. Another nested bind is present that performs a similar task starting with values of d_dice2, eventually producing a distribution, d_sum over the sum of two dice rolls. The probability of this sum being 10 is looked up at the last line.

To use this approach, functions must be rewritten in monadic style. For example, the rewritten rand_strat function (given just below Definition 3, page 6) is the following:

```
let rand_strat
  (t: time) (lows: ℒ list) (obss: 𝒪 list): 𝒜 dist =
  bind_flip 0.5
    (fun flip ->
      if flip then
        bind_uniform_select ℰlist
          (fun exp -> return (Attack exp))
      else
        bind_uniform_select ℒlist
          (fun low -> return (Wait low)))
```

Notice that the output type is now a distribution on actions, not just a single action. Monadic style becomes cumbersome with more complex functions, and though a more direct-style implementation is possible (e.g., as in the second half of Kiselyov and Shan's paper [31]), we found it did not perform as well.

Our current implementation makes little attempt to optimize the construction of the distribution, and thus is susceptible to a state-space explosion when a scenario has many time steps. Fortunately, probabilistic programming is a growing research area (cf. Gordon et al's survey [32]) so we are optimistic that the scale of experiments we can consider will increase in the future. Approximate probabilistic inference systems such as those based on graphical models or sampling can be used to estimate gain. Exact implementations based on smarter representations of distributions such as those using algebraic decision diagrams [32] could potentially used to simulate our model. Additionally, Mardziel et al. [9], [33] show how to soundly approximate a simple metric using probabilistic abstract interpretation; this technique could potentially be extended to soundly approximate dynamic gain. Presently, our simple implementation proved to be sufficient to demonstrate various aspects of the model on a variety of scenarios.

VI. Experiments

This section describes several experiments we conducted, illustrating the interesting outcomes mentioned in the introduction. In Section VI-B we show how the addition of dynamic secrets impacts the existing metrics we expressed as dynamic gain. It can be seen there that the uncertainty about the secret can be effectively recovered with time. On the other hand, in Section VI-D we show that if an adversary can wait, their gain can only increase with time. Additionally we show in Section VI-C that low adaptive adversaries can be significantly more dangerous (even exponentially so) than those without low choices. In Section VI-E we show how dynamic gain can be effectively bounded even with fully adaptive adversaries by asserting a cost for each observation. This example also shows that our model might not always be fitting in situations where adversary's and system's goals are not necessarily zero-sum. Finally, in Section VI-F we show that it is not always better to change the secret more often.

A. Setting: Stakeouts and raids

We will demonstrate our model and various existing and extended security metrics using several examples on the theme of *stakeouts and raids*. We describe the common elements of the scenario here, and consider the effect of varying them in the following subsections.

Suppose an illicit substance dealer is locked in an ever-persistent game of hiding his stash from the police. At its simplest, the example is identical to that of password guessing, replacing the password with the location of the stash and authentication attempts with "stakeouts" in which police observe a potential stash location for the presence of the stash. After making observations the police will have a chance to "raid" the stash, potentially succeeding. In the meantime the stash location might change.

The stash location will be represented as a integer from 0 to 7, as will be the attacks. Observations will be booleans:

```
type ℋ = ℒ = ℰ = int                        (* 0,...,7 *)
type 𝒪 = bool
```

Stakeouts are carried out by comparing the stakeout location (according to a fixed ordering over time) to the real stash location.

```
let stakeout_rotate: sysf =
  fun (hist: history) ->
    last hist.highs = (hist.t mod 8)
```

Note that this is the *system* function (not the strategy function), and it provides observations to the (optimal) adversary in a fixed order; low inputs in the history are ignored. The fixed stakeout order is used in most experiments as there is no benefit in picking stakeout locations in a specific order over any other order (or picking it adaptively). We will use a different system function for stakeouts in Section VI-C to demonstrate the power of low-adaptivity.

Raids fail unless it is the time step T and the stash location is equal to the raid location:

```
let raid_non_adapt: gainf =
  fun (hist: history) (exp: ℰ) ->
    if hist.t = T
      then if last hist.highs = exp then 1.0 else 0.0
      else −∞
```

The failure at any time other than `T` ensures that we only consider adversaries that cannot adaptively decide when to attack. In Section VI-D we will use a wait-adaptive version of this attack function which will be missing this restriction.

Finally, for most experiments, the stash moves randomly every 4 time steps (`rate` is 4):

```
let move_stash (rate: int): highf =
  fun (hist: history) ->
    if hist.t mod rate = 0
      then gen_stash ()
      else last hist.highs
```

Above, we reference the `gen_stash` function introduced earlier, which randomly selects a stash location.

```
let gen_stash () =
  let real_loc = (random_int () mod 8) in real_loc
```

The change for our final example (Section VI-F) is more complex and will be described later in the paper.

B. How does gain differ for dynamic secrets, rather than static ones?

Our first experiment considers the impact on information leakage when the secret can change. For comparison, we also consider a constant high-input generation function:

```
let stay_stash: highf =
  fun (hist: history) ->
    if hist.t = 0 then gen_stash ()
                  else last hist.highs
```

We also consider a variation of a non-adaptive raid that has a chance to fail even if the raid location is correct, and has a chance to accidentally discover a new stash even if raid takes place at the wrong location. This is meant only to demonstrate a very simple attack function that is not identical to guessing the secret.

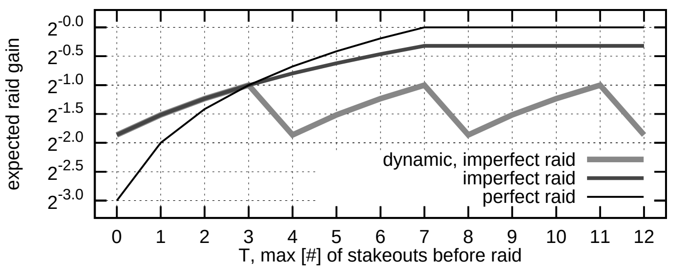


Fig. 5. Expected raid gain over time given stakeouts `stakeout_rotate` with (1) static stash `stay_stash` and perfect raids `raid_non_adapt`, (2) static stash and imperfect raids `raid_non_adapt_imperfect`, and (3) moving stash `move_stash 4` and imperfect raids.

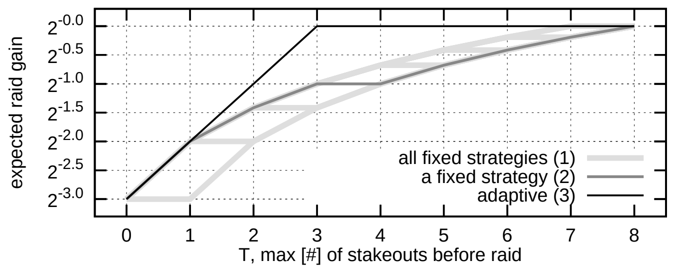


Fig. 6. Expected raid `raid_non_adapt` gain over time with static stash `stay_stash` given stakeouts with (1) all possible non-adaptive orderings of `stakeout_east_west_fixed`, (2) a possible non-adaptive ordering of `stakeout_east_west_fixed`, and (3) adaptive `stakeout_east_west` stakeout locations.

```
let raid_non_adapt_imperfect: gainf =
  fun (hist: history) (exp: ℰ) ->
    if hist.t = T
      then if last hist.highs = exp
        then flip 0.8
        else flip 0.2
      else −∞
```

Figure 5 summarizes the general behavior of this example. Demonstrated are cases where change does not occur, raids are imperfect, and the later with changing stash location. The non-dynamic portion (1) with perfect raid is an example of an analysis achievable by a parallel composition of channels and the min-vulnerability metric. Adding the imperfect attack function (2) alters the shape of vulnerability over time though in a manner that is *not a mere scaling* of the perfect attack case. The small chance of a successful raid at the wrong location results in higher gains (compared to perfect raid) when knowledge is low. With more knowledge, the perfect raid results in more gain than the imperfect. Adding a dynamically changing stash (3) results in a periodic, non-monotonic, gain; though gain increases in the period of unchanging secret, it falls right after the secret changes. This, in effect, is a recovery of uncertainty which is missing in existing information flow literature, where uncertainty is seen as a non-renewable resource [34]. In the following sections we will refer to the period of time in which the secret does not change as an *epoch*.

C. How does low-adaptivity impact information leakage?

To demonstrate the power of low-adaptivity we will use a system function that outputs whether the stash is east or west of

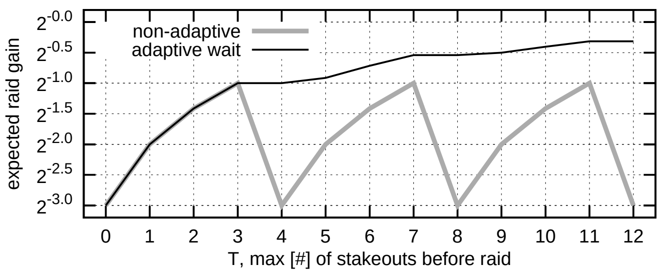


Fig. 7. Expected raid gain with moving stash `move_stash 4` given stakeouts `stakeout_rotate` with (1) non-adaptive `raid_non_adapt` and (2) wait-adaptive raids `raid_wait_adapt`.

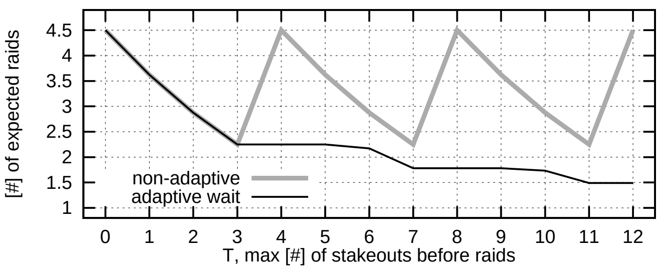


Fig. 8. Expected number of raids over with moving stash `move_stash 4` given stakeouts `stakeout_rotate` with (1) non-adaptive `raid_guess_non_adapt` and (2) wait-adaptive raids `raid_guess_wait_adapt`.

the stakeout location. Assuming the stash locations are ordered longitudinally, this function is just a comparison between the stash and stakeout location. The non-adaptive adversary will pick the stakeouts in a fixed order specified by the variable `order` whereas the adaptive adversary will use the low input:

```
let stakeout_east_west_fixed (order: ℒ list): sysf =
  fun (hist: history) ->
    last hist.highs <= order.(hist.t)
```

```
let stakeout_east_west: sysf
  fun (hist: history) ->
    last hist.highs <= last hist.lows
```

Notice how the second function refers to the most recent low input, while the first ignores low inputs entirely.

Figure 6 demonstrates the expected gain of both types of attackers. For fixed-order attackers, we used all $8!$ permutations of the 8 stash locations as possible orders and plotted them all as the wide light gray lines in the figure. Though there are many possible orders, the only thing that makes any difference in the gain over time is the position of 7 (the highest stash location) in the ordering as the system function for this input reveals no information whatsoever. All other stakeout locations reveal an equal amount of information in terms of the expected gain. To demonstrate this behavior, we have specifically plotted in the figure the gain for an ordering in which location 7 is staked-out at time 3 (labeled "a fixed strategy"). Gain increases linearly with every non-useless observation. On the other hand, the optimal low-adaptive adversary performs binary search, increasing his gain exponentially with time.

D. How does wait-adaptivity impact information leakage?

An adversary that can wait is allowed to attack at any time, and we enable the optimal adversary to do so by adjusting the gain function:

```
let raid_wait_adapt: gainf =
  fun (hist: history) (exp: ℰ) ->
    if last hist.highs = exp then 1.0 else 0.0
```

Adaptive wait has a significant impact on the success an adversary might expect. In the simple stakeout/raid example of Figure 7, it transforms the an ever-bounded vulnerability to one that steadily increases in time. Conversely, as shown in Figure 8, the guessing entropy steadily decreases (recall guessing entropy is inversely related to gain in our model); this experiment uses the following variants of the gain functions:

```
type ℰ = ℋ list

let raid_guess_wait_adapt: gainf =
  fun (hist: history) (exp: ℰ) ->
    -1.0*(1.0+(pos_of (last hist.highs) exp))
```

```
let raid_guess_non_adapt: gainf =
  fun (hist: history) (exp: ℰ) ->
    if hist.t < T then −∞
      else raid_guess_wait_adapt hist exp
```

The gain over time for an optimal wait-adaptive adversary in these examples is as follows. Roughly, the optimal behavior is to wait until a successful stakeout before attacking. The more observations there are, the higher the chance this will occur. This results in the monotonic trend in gain over time.

Additionally there are subtle decisions the optimal adversary makes in order to determine whether to wait and allow the secret to change. For example, in Figure 7, if the adversary has to attack at time 5 or earlier and has not yet observed a successful stakeout by time 3, they will wait until time 5 to attack, letting all their accumulated knowledge be invalidated by the change that occurs at time 4. This seems counter-intuitive as their odds of a successful raid at time 3 is $1/5$ (they eliminated 3 stash locations from consideration), whereas they will accumulate only 1 relevant stakeout observation by time 5. Having 3 observations seems preferable to 1. The optimal adversary will wait because the *expected* gain at time 5 is actually better than $1/5$; there is $1/8$ chance that the stakeout at time 5 will pinpoint the stash, resulting in gain of 1, and $7/8$ chance it will not, resulting in expected gain of $1/7$. The expectation of gain is thus ${}^1\!/\!_8 \cdot 1 + {}^7\!/\!_8 \cdot {}^1\!/\!_7 = {}^1\!/\!_4 > {}^1\!/\!_5$. The adversary thus has better expected gain if they have to attack by time 5 as compared to having to attack by time 3 or 4, despite having to raid with only one observation's worth of knowledge. One can modify the parameters of this experiment so that this does not occur, forcing the adversary to attack before the secret changes. This results in a longer period of constant vulnerability after each stash movement.

This ability to wait is the antithesis of moving-target vulnerability. Though a secret that is changing with time serves to keep the vulnerability at any fixed point low, the vulnerability *for some point* can only increase with time. The drug dealer of our running example would be foolhardy to believe that he is safe from a police raid just because it is unlikely to happen on Wednesday, or any other fixed day; if stakeouts continue and the police are smart enough to not

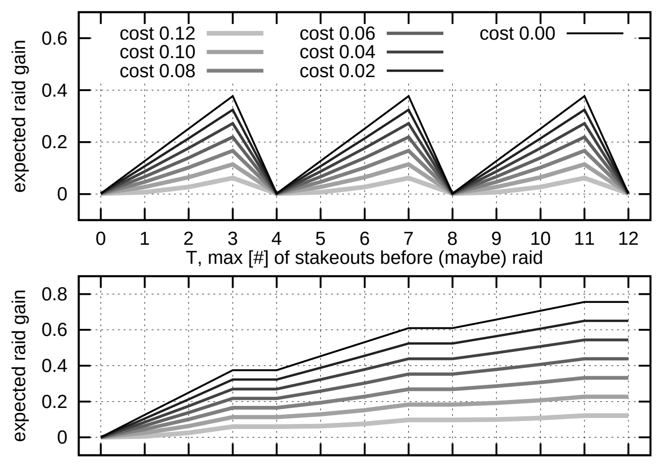


Fig. 9. Expected raid gain with costly stakeouts `stakeout_option` and moving stash `move_stash 4` with (top) non-adaptive `raid_option_non_adapt` and (bottom) wait-adaptive raids `raid_option`.

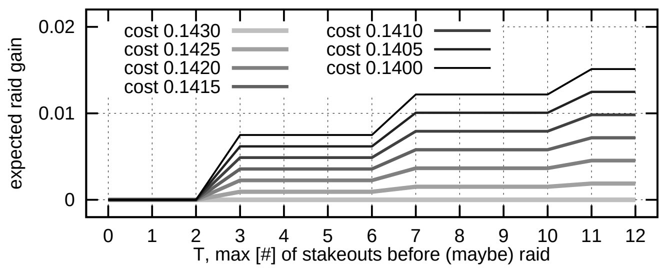


Fig. 10. (Zoomed in) expected raid gain with costly stakeouts `stakeout_option` and moving stash `move_stash 4` and wait-adaptive raids `raid_option` for high stakeout costs.

schedule drug busts before having performed the surveillance, the dealer will be caught. On the other hand, if there is a high-enough cost associated with making an observation, the vulnerability against raids can be very effectively bounded (as we show in the next experiment).

In fact, we can prove that the monotonically increasing vulnerability is a property of any scenario where the adversary can decide when to attack.

Theorem 7. *Given any gain function* `g` *that is invariant in* T *(the value* `hist.tmax`*), we have* $\mathbb{D}_g($`evaluate (t+1) ...`$) \geq \mathbb{D}_g($`evaluate t ...`$)$.

Proof: (Sketch) The theorem holds as an adversary attacking a system with $T = t{+}1$ will have a chance to attack at time t, using the same exact gain function that the adversary would attack were $T = t$, and using the same inputs. We assumed the gain functions are invariant in T hence their gains must be identical. Naturally in the first situation, the attacker can also wait if the expectation of gain due to waiting one more time step is higher. ■

E. Can gain be bounded by costly observations?

Our model is general enough to express costs associated with observations. For example, we can modify our scenario so that the police either observe nothing (indicated by low input and output `None`), or perform a stakeout.

```
type ℒ = ℰ = int option         (* int in 0,...,7 *)
type 𝒪 = bool option
```

```
let stakeout_option: sysf =
  fun (hist: history) ->
    match last hist.lows with
     | None -> None
     | Some stakeout ->
       Some (last hist.highs = stakeout)
```

However, each stakeout performed will have a cost that is applied to the final gain (`c` per stakeout) Additionally, the police are penalized -1.0 units for a raid on the wrong place, and have the option of not raiding at all (for 0.0 gain).

```
let raid_option (cost: float): gainf =
  fun (hist: history) (exp: ℰ) ->
    let raid_gain =
      match exp with
        | None -> 0.0
        | Some raid_loc ->
          if last hist.highs = raid_loc
            then 1.0
            else -1.0 in
    let stakeouts = (count_some hist.lows) in
        (* count how many low inputs in
           hist.lows were not None *)
      raid_gain - cost * stakeouts
```

We also define a non-wait-adaptive version of this function, for purposes of comparison.

```
let raid_option_non_adapt (cost: float): gainf =
  fun (hist: history) (exp: ℰ) ->
    if hist.t < tmax then −∞
                     else raid_option cost hist atk
```

The results of this scenario are summarized in Figure 9 for stakeout costs ranging from 0.00 to 0.12 and in Figure 10 for higher costs in the range 0.1400 to 0.1430. In the top half of the first figure, the police do not have a choice of when to attack and the optimal behavior is to only perform stakeouts in the epoch that the attack will happen. Higher costs scale down the expected gain. This results in the periodic behavior seen in the figure.

For wait-adaptive adversaries the result is more interesting. For sufficiently small stakeout costs, the optimal adversary will keep performing stakeouts at all times except for the time period right before the change in the stash location and attack only when the stash is pinpointed. This results in the temporary plateau every 4 steps seen in the bottom half of Figure 9. This behavior seems counter-intuitive as one would think the stakeout costs will eventually make observations prohibitively expensive. Note, however, that every epoch is identical in this scenario, the stash location is uniform in $0, ..., 7$ at the start, and the adversary has 4 observations before it gets reset. If the optimal behavior of the adversary in the first epoch is to stakeout (at most 3 times), then it is also optimal for them to do it during the second (if they have not yet pinpointed the stash). It is still optimal on the $(n{+}1)^{th}$ epoch after failures in the first n. As it is, the expectation of gain due to 3 stakeouts is higher than no guesses in any epoch, despite the costs.

The optimal behavior is slightly different for high enough stakeout costs but the pattern remains the same. Figure 10

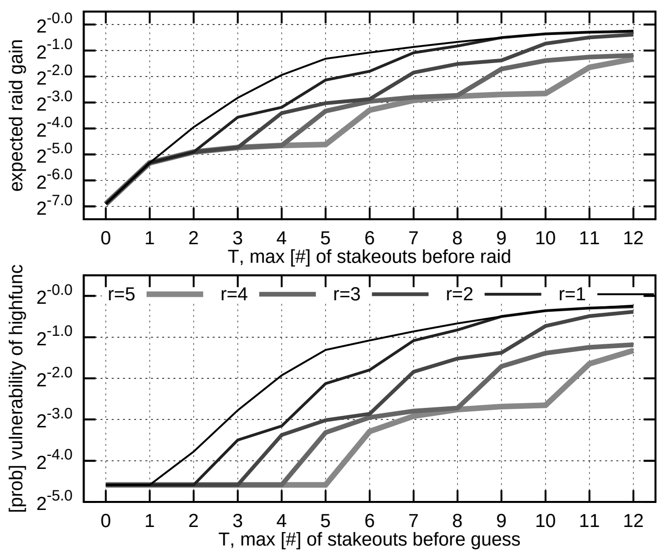


Fig. 11. Changing stash according to `gangs_move` with stakeouts `stakeout_building` quantifying (top) expected raid `raid_apartment` gain and (bottom) expected `high_func` vulnerability.

shows the result of an adversary staking out in an epoch only if he is allowed enough time to attack at the end of the epoch. That is, for T equal to 1 or 2, the police would not stakeout at all, but if T is 3, they will stakeout at times 1,2, and 3. This is because the expected gain given 1 or 2 stakeouts is lower than 0, but for 3 stakeouts, it is greater than 0. As was the case with the lower cost, since the expected gain of observing in an epoch (3 times) is greater than not, the optimal adversary will continue to observe indefinitely if he is given the time.

Note however, that observing indefinitely in these examples does not mean that the expected gain approaches 1. Analytical analysis of this scenario tells us that after n full epochs, the expected gain is $1/8(3-21c)\sum_{i=0}^{n-1}(5/8)^i$ and in the limit, this quantity approaches $1-7c$. The adversary's gain is thus bounded by $1-7c$ for varying stakeout costs c (the point at which it is optimal to not observe at all is $c=1/7\approx 0.1429$).

The fact that the adversary's gain can be bounded arbitrarily close to 0, despite their strategy of performing stakeouts indefinitely, highlights a shortcoming of our present model: the assumption of zero-sum relationship between adversary and system. By quantifying optimal adversary's gain, we are implicitly assuming that their gain is our loss. This, however, is hard to justify in some situations like the example of this section. Under the optimal adversary, the drug dealer's stash will be discovered despite the bounded expected adversary gain from said discovery. Ideally the drug dealer's gain should be a different function than the negation of the police's gain. This idea, and the more game theoretic scenario it suggests, forms part of our ongoing work.

F. Does more frequent change necessarily imply less gain?

In our last example, we will use an as-yet unused feature of our model to demonstrate a counter-intuitive fact that more change is not always better. So far we have used a high-input generation function (or "change function", for short) that is known to the attacker but here he will only know the set of possible change functions. Furthermore, guessing the full secret will *require* knowledge of the change function and therefore will require change to occur sufficiently many times.

Let `nbuilds` be the number of buildings (in the example figure, `nbuilds` will be 5), and `nfloors = factorial (nbuilds-1)` be the number of floors in each of these buildings. These (`nbuilds`) shady buildings are in a city where illicit stashes tend to be found. Each building has `nfloors` floors and each floor of each building is claimed by a drug-dealing gang. A single gang has the same numbered floor in all the buildings. That is, gang 0 will have floor 0 claimed in every building, gang 1 will have floor 1 in every building, and so on.

The police know there is a stash hidden on some floor in some building and that every gang moves their stash once in a while from one building to another in a predictable pattern (the floor doesn't change). They know all `nbuilds` gangs each have a unique permutation π of $0,\ldots,$`nbuilds-1` as their stash movement pattern. The police also know which floors belong to which gangs (and their permutation). Given `r` as the stash movement rate parameter, the movement of the stash is governed by the following function:

```
type ℋ = {building: int; floor: int}
   (* building int in 0,...,4 *)
   (* floor int in 0,...,23 *)

let gang_move: highf =
  let gang = (random_int ()) mod nfloors in
  let π = (gen_all_permutations nbuilds).(gang) in

  fun (hist: history) ->
    let c_floor = (last hist.highs).floor in
    let c_build = (last hist.highs).building in
    if hist.t > 0
    then begin
      if t mod r = 0
        then {building = π (c_build);
              floor    = c_floor}
        else stash
    end else
      { building = (random_int ()) mod nbuilds;
        floor    = c_floor }

    in gang_move
```

The function first picks a random gang and generates the permutation that represents that gang's stash movement pattern. It then creates a change function that will perform that movement, keeping the floor the same, while moving from building to building (the function picks a random building at time 0).

The police set up stakeouts to observe all the buildings but are only successful at detecting activity half the time, and they cannot tell on which floor the stash activity takes place, just which building:

```
type 𝒪 = int option           (* int in 0,...,4 *)

let stakeout_building: sysf =
  fun (hist: history) ->
    if flip 0.5
      then Some (last hist.highs).building
      else None
```

The police want to raid the stash but cannot get a warrant for the whole building, they need to know the floor too:

```
type ℰ = ℋ

let raid_apartment: gainf =
  fun (hist: history) (exp: ℰ) ->
    let build = (last hist.highs).building in
    let floor = (last hist.highs).floor in
    if build = exp.building &&
       floor = exp.floor
    then 1.0 else 0.0
```

Now, the chances of a successful police raid after a varying number of stakeouts depends on the stash change rate r. Unintuitively, frequent stash changes lead the police to the stash more quickly. Figure 11(top) shows the gain in the raid after various number of stakeouts, for four different stash change rates (nbuilds = 5). The chances of a failed observation in this example are not important and are used to demonstrate the trend in gain over a longer period of time. Without this randomness, the gains quickly reach 1.0 after $r * (\texttt{nbuilds} - 1)$ observations (exactly enough to learn the initially unknown permutation).

The example has a property that the change function (the permutation π of the gang) needs to be learned in order to determine the floor of the stash accurately. Observing infinitely many stakeouts of the same building would not improve police's chance beyond 1 in nfloors; learning the change function here absolutely requires learning how it changes the secret. One can see the expected progress in learning the change function in Figure 11(bottom), note the clear association between knowing the change function and knowing the secret. We theorize that this correlation is a necessary part of examples that have the undesirable property that more change leads to more vulnerability. Characterizing this for scenarios in general is another part of our ongoing work.

VII. Related Work

Other works in the literature have considered systems with some notion of time-passing. Massey [35] considers systems that can be re-executed several times, whereby new secret and observable values are produced constantly. He conjectured that the flow of information in these systems is more precisely quantified by *directed information*, a form of Shannon entropy that takes causality into consideration, which was later proved correct by Tatikonda and Mitter [28]. Alvim et al. use these works to build a model for *interactive systems* [27], in which secrets and observables may interleave and influence each other during an execution. The main differences between their model and ours are: (i) they see the secret growing with time, rather than evolving; (ii) they consider Shannon-entropy as a metric of information, rather than vulnerability metrics; and (iii) they only consider passive adversaries.

Köpf and Basin [15] propose a model for adaptive attacks on deterministic systems, and show how to calculate bounds on the information leakage of such systems. Our model generalizes theirs in that we consider probabilistic systems. Moreover, we distinguish between the adversarial production of low inputs and of exploits, and allow adversaries to wait until the best time to attack, based on observations of the system, which itself could be influenced by the choice of low inputs.

In the context of Location Based Services, the privacy of a moving individual is closely related to the amount of time he spends in a certain area, and Marconi et al. [36] demonstrate how an adversary with structured knowledge about a user's behavior across time can pose a direct threat to his privacy.

The work of Shokri et al. [37] strives to quantify the privacy of users of location based services using Markov models and various machine learning techniques for constructing and applying them. Location privacy is a useful application of our framework, as a principal's location may be private, and evolves over time in potentially predictable ways. Shokri et al.'s work employs two phases, one for learning a model of how a principal's location could change over time, and one for de-anonymizing subsequently observed, but obfuscated, location information using this model. Our work focuses on information theoretic characterizations of security in such applications, and permits learning the change function and the secrets in a continual, interleaved (and optimized) fashion. That said, Shokri et al's simpler model allows them to consider more realistic examples than those described in our work. Subsequent work [38] considers even simpler models (e.g., that a user's locations are independent).

Some approaches model interactivity by encoding it as a single "batch job" execution of the system. Desharnais et al. [26], for instance, model the system as a channel matrix of conditional probabilities of whole output traces given whole input traces. Besides creating technical difficulties for computing maximum leakage [27], this approach does not permit low-adaptive or wait-adaptive adversaries, because it lacks the feedback loop present in our model.

O'Neill et al. [13], based on Wittbold and Johnson [11], improve on batch-job models by introducing strategies. The strategy functions of O'Neill et al. are deterministic, whereas our are probabilistic. And their model does not support wait-adaptive adversaries. So our model of interactivity subsumes theirs.

Clark and Hunt [12] present two models of interactivity. In one model, agents provide all their inputs before program execution as a stream of values. That model does not support low-adaptive adversaries, whereas our model in Section III does. In Clark and Hunt's other model, agents use strategy functions to provide inputs. Their model's strategy functions are nondeterministic, whereas our model's are probabilistic. Probabilistic choice refines nondeterministic choice [39], so in that sense our model is a refinement of Clark and Hunt's. Importantly, neither of Clark and Hunt's models permits wait-adaptive adversaries. There are other nonessential differences between the models: Clark and Hunt allow a lattice of security levels, whereas we allow just high and low; our model only produces low outputs, whereas theirs permits outputs at any security level; and our computation model is probabilistic automata, whereas theirs is labeled transition systems.

VIII. Conclusions and Future Work

In this paper we have presented a new model for quantifying the information flow of a system whose secrets evolve over time. Our model involves an adaptive adversary, and characterizes the costs and benefits of attacks on the system. We showed that an adaptive view of an adversary is crucial in calculating a system's true vulnerability which could be greatly underestimated otherwise. We also showed that though adversary uncertainty can effectively be recovered if the secret changes, if adversary can adaptively wait to attack, vulnerability can only increase in time. Also, contrary to intuition, we showed that more frequent changes to secrets can actually make them more vulnerable.

Our future work has three main thrusts. Firstly we are generalizing our model further to give the user non-fixed decisions and goals distinct from those of the adversary. Doing so will let us better model examples like that of Section VI-E where adversary gain does not quite mirror the loss of the user. Furthermore, to handle scenarios in which the user and adversary do not fully observe each other's decisions, or when they make decisions simultaneously, we are looking into a game-theoretic analysis of the problem using Bayesian games and their equilibria.

The second avenue of our future work is information theoretical characterizations of various phenomena present in our model which we have hinted at in this paper:

• In Section VI-C we saw that the impact of low-adaptive adversaries range from none to an exponential difference in gain. It is easier, however, to analyze non-adaptive adversaries. It would be valuable to be able to tell when ignoring low-adaptivity will not have a significant impact on the resulting analysis.

• In Section VI-F we saw how changing the secret more often is not always preferable to changing it less. We conjectured that such situations require a strong correlation between the secret and the high generation function used to evolve the secret. Precisely characterizing this correlation and the contexts in which it is relevant would be useful for building more robust systems.

• The analyses in the paper are framed in the context of a system. Unfortunately, this context directly influences how much information is leaked, so that the less that is known about the context, the less we can say about the security of the system. Prior works attempt to speak of the worst-case leakage for all possible contexts, but for us contexts are richer in structure, making such analysis more difficult. We consider such worst-case reasoning challenging future work.

Finally we are bringing in the works on approximate but sound probabilistic programming [9], [33] to enable the simulation of larger and more complex scenarios. Though these works are only concerned with a metric similar to min-vulnerability, it may be possible to extend the approach to soundly approximate dynamic gain as well. Additionally exact probabilistic programming systems with smarter representations of distributions such as algebraic decision diagrams in [32] could potentially be applied to our model. We are also investigating this possibility.

References

[1] I. S. Moskowitz, R. E. Newman, and P. F. Syverson, "Quasi-anonymous channels," in *Proc. of CNIS*. IASTED, 2003, pp. 126–131.

[2] D. Clark, S. Hunt, and P. Malacaria, "Quantitative information flow, relations and polymorphic types," *J. of Logic and Computation*, vol. 18, no. 2, pp. 181–199, 2005.

[3] K. Chatzikokolakis, C. Palamidessi, and P. Panangaden, "Anonymity protocols as noisy channels," *Inf. and Comp.*, vol. 206, no. 2–4, pp. 378–401, 2008. [Online]. Available: http://hal.inria.fr/inria-00349225/en/

[4] G. Smith, "On the foundations of quantitative information flow," in *Proceedings of the Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, 2009.

[5] M. R. Clarkson, A. C. Myers, and F. B. Schneider, "Quantifying information flow with beliefs," *Journal of Computer Security*, vol. 17, no. 5, pp. 655–701, 2009.

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

[7] 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.

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

[9] P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa, "Dynamic enforcement of knowledge-based security policies," in *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2011.

[10] R. Segala, "Modeling and verification of randomized distributed real-time systems," Ph.D. dissertation, Massachusetts Institute of Technology, Jun. 1995, tech. Rep. MIT/LCS/TR-676.

[11] J. T. Wittbold and D. Johnson, "Information flow in nondeterministic systems," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 1990, pp. 144–161.

[12] D. Clark and S. Hunt, "Non-interference for deterministic interactive programs," in *Proceedings of the Workshop on Formal Aspects in Security and Trust*, 2008, pp. 50–66.

[13] K. R. O'Neill, M. R. Clarkson, and S. Chong, "Information-flow security for interactive programs," in *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2006, pp. 190–201.

[14] M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith, "Measuring information leakage using generalized gain functions," in *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2012.

[15] B. Köpf and D. Basin, "An information-theoretic model for adaptive side-channel attacks," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[16] D. Denning, *Cryptography and Data Security*. Reading, Massachusetts: Addison-Wesley, 1982.

[17] J. Y. Halpern, *Reasoning about Uncertainty*. Cambridge, Massachusetts: MIT Press, 2003.

[18] P. Malacaria, "Assessing security threats of looping constructs," in *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, M. Hofmann and M. Felleisen, Eds. ACM, 2007, pp. 225–235. [Online]. Available: http://doi.acm.org/10.1145/1190216.1190251

[19] P. Malacaria and H. Chen, "Lagrange multipliers and maximum information leakage in different observational models," in *Proceedings of the 2008 Workshop on Programming Languages and Analysis for Security (PLAS 2008)*, Úlfar Erlingsson and Marco Pistoia, Ed. Tucson, AZ, USA: ACM, June 2008, pp. 135–146.

[20] I. S. Moskowitz, R. E. Newman, D. P. Crepeau, and A. R. Miller, "Covert channels and anonymizing networks." in *Workshop on Privacy in the Electronic Society 2003*, 2003, pp. 79–88.

[21] M. S. Alvim, M. E. Andrés, and C. Palamidessi, "Information Flow in Interactive Systems," in *Proceedings of the 21th International Conference on Concurrency Theory (CONCUR 2010), Paris, France, August 31-September 3*, ser. Lecture Notes in Computer Science, P. Gastin and F. Laroussinie, Eds., vol. 6269. Springer, 2010, pp. 102–116. [Online]. Available: http://hal.archives-ouvertes.fr/inria-00479672/en/

[22] Massey, "Guessing and entropy," in *Proceedings of the IEEE International Symposium on Information Theory*. IEEE, 1994, p. 204.

[23] P. Malacaria, “Algebraic foundations for information theoretical, probabilistic and guessability measures of information flow,” *CoRR*, vol. abs/1101.3453, 2011.

[24] Pliam, “On the incomparability of entropy and marginal guesswork in brute-force attacks,” in *Proceedings of INDOCRYPT: International Conference in Cryptology in India*, ser. Lecture Notes in Computer Science, no. 1977. Springer-Verlag, 2000, pp. 67–79.

[25] C. Braun, K. Chatzikokolakis, and C. Palamidessi, “Quantitative notions of leakage for one-try attacks,” in *Proceedings of the 25th Conf. on Mathematical Foundations of Programming Semantics*, ser. Electronic Notes in Theoretical Computer Science, vol. 249. Elsevier B.V., 2009, pp. 75–91. [Online]. Available: http://hal.archives-ouvertes.fr/inria-00424852/en/

[26] J. Desharnais, R. Jagadeesan, V. Gupta, and P. Panangaden, “The metric analogue of weak bisimulation for probabilistic processes,” in *LICS*, 2002, pp. 413–422.

[27] M. S. Alvim, M. E. Andrés, and C. Palamidessi, “Quantitative information flow in interactive systems,” *Journal of Computer Security*, vol. 20, no. 1, pp. 3–50, 2012.

[28] S. Tatikonda and S. K. Mitter, “The capacity of channels with feedback,” *IEEE Transactions on Information Theory*, vol. 55, no. 1, pp. 323–349, 2009.

[29] “The Caml language,” http://caml.inria.fr.

[30] N. Ramsey and A. Pfeffer, “Stochastic lambda calculus and monads of probability distributions,” in *Proceedings of the ACM SIGPLAN Conference on Principles of Programming Languages (POPL)*, 2002.

[31] O. Kiselyov and C. chieh Shan, “Embedded probabilistic programming,” in *Proceedings of the Working Conference on Domain Specific Languages (DSL)*, 2009.

[32] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani, “Probabilistic programming,” in *International Conference on Software Engineering (ICSE, FOSE track)*, 2014. [Online]. Available: http://research.microsoft.com/pubs/208585/fose-icse2014.pdf

[33] P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa, “Dynamic enforcement of knowledge-based security policies using abstract interpretation,” *Journal of Computer Security*, vol. 21, no. 4, pp. 463–532, 2013.

[34] B. Espinoza and G. Smith, “Min-entropy as a resource,” in *Information and Computation*, 2013.

[35] J. L. Massey, “Causality, feedback and directed information,” in *Proc. of the 1990 Intl. Symposium on Information Theory and its Applications*, November 1990.

[36] L. Marconi, R. D. Pietro, B. Crispo, and M. Conti, “Time warp: How time affects privacy in lbss,” in *ICICS*, 2010, pp. 325–339.

[37] R. Shokri, G. Theodorakopoulos, J.-Y. L. Boudec, and J.-P. Hubaux, “Quantifying location privacy,” in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2011.

[38] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. L. Boudec, “Protecting location privacy: optimal strategy against localization attacks,” in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 617–627.

[39] A. McIver and C. Morgan, *Abstraction, Refinement and Proof for Probabilistic Systems*. New York: Springer, 2005.

Appendix

A. On mathematical soundness of the model

The general behavior of the system after T time steps can be completely described by the joint probability distribution $\Pr(h^T, \ell^T, o^T)$ derived from the execution of the probabilistic automaton representing the system:

$$\begin{aligned}\Pr(h^T, \ell^T, o^T) = \prod_{t=1}^{T} [\ & \Pr(h_t \mid h^{t-1}, \ell^{t-1}, o^{t-1}) \\ & \cdot \Pr(\ell_t \mid \ell^{t-1}, o^{t-1}) \\ & \cdot \Pr(\ell_t \mid h^t, \ell^t, o^{t-1})\ ]\end{aligned} \tag{6}$$

The distribution p, however, is defined over the history of high inputs, low inputs and observables only, and does not explicitly model the way the context generate high inputs and low inputs. For this we define deterministic high-input strategy functions and deterministic strategy functions, respectively. By endowing these functions with probability distributions we can emulate the probabilistic occurrence of high inputs and low inputs derived from the probabilistic automaton. The context of the system execution is described, thus, by a distribution $\{\Pr(\eta_t \mid \eta^{t-1})\}_{t=1}^T$ on high-input strategy functions and a distribution $\{\Pr(\alpha_t \mid \alpha^{t-1})\}_{t=1}^T$ on strategy functions.

We bring the system and its contexts together using a technique proposed by Tatikonda and Mitter [28], and applied by Alvim et al. to interactive systems [27]. The technique lifts the core-channel—i.e., the box named “system” in Figure 2, which takes high and low inputs and produces observables—to an equivalent channel whose inputs are high-input generation functions and strategy functions. This new channel, represented as the outer dashed-box in Figure 2, models the full scenario of an execution. At each time step t a high-input strategy function η_t and an strategy function α_t are fed to the channel, and an observable o_t is produced. This channel presents no feedback to its inputs, avoiding thus some technical problems that would arise from deriving a channel directly from (6). [11]

To completely describe the behavior of the full scenario after T time steps we define a joint probability distribution $Q(\eta^T, \alpha^T, h^T, \ell^T, o^T)$ that extends (6) to also include random variables modeling high-input strategy functions and strategy functions. The distribution Q is said to be *consistent* with respect to a scenario if it appropriately captures the inter-relation among the context and the system in Figure 2.

Definition 8. A measure $Q(\eta^T, \alpha^T, h^T, \ell^T, o^T)$ is *consistent* with respect to the probability distribution on high-input strategy functions $\{\Pr(\hat{\eta}_t \mid \hat{\eta}^{t-1})\}_{t=1}^T$, to the probability distribution on strategy functions $\{\Pr(\hat{\alpha}_t \mid \hat{\alpha}^{t-1})\}_{t=1}^T$, and to the core-channel $\{\Pr(\mathcal{O}_t \mid \mathcal{H}^t, \mathcal{L}^t, \mathcal{O}^{t-1})\}_{t=1}^T$ if, for each t:

1) There is no feedback to the high-input strategy function: $Q(\eta_t \mid \eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) = \Pr(\eta_t \mid \eta^{t-1})$.

2) There is no feedback to the low-input strategy function: $Q(\alpha_t \mid \eta^t, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) = \Pr(\alpha_t \mid \alpha^{t-1})$.

3) The high input is a function of the past high inputs, low inputs, and observable outputs: $Q(h_t \mid \eta^t, \alpha^t, h^{t-1}, \ell^{t-1}, o^{t-1}) = \delta_{\{\eta_t(h^{t-1}, \ell^{t-1}, o^{t-1})\}}(h_t)$, where δ is the Dirac measure.

4) The low input is a function of the past low inputs, and observable outputs: $Q(\ell_t \mid \eta^t, \alpha^t, h^t, \ell^{t-1}, o^{t-1}) = \delta_{\{\alpha_t(\ell^{t-1}, o^{t-1})\}}(\ell_t)$, where δ is the Dirac measure.

5) The probabilistic mapping from high inputs and low inputs to observables is preserved as in the original system: $Q(o_t \mid \eta^t, \alpha^t, h^t, \ell^t, o^{t-1}) = \Pr(o_t \mid h^t, \ell^t, o^{t-1})$.

[11] As shown in Alvim et al. [27], in the presence of *feedback* (that is, when the generation of high inputs and low inputs are not independent from the observables produced by the system), (6) does not specify a channel that is invariant with respect to the distribution on low and high inputs. This may be a problem when deriving maximum bounds for the information leakage of a system, as traditional information-theoretic approaches need the the channel $\{\Pr(o_t \mid h^t, \ell^t, o^{t-1})\}_{t=1}^T$ to be invariant.

We now show the model of Figure 2 is well-suited for the systems we are interested in. First, Theorem 1 shows that, given a description of a system and of a context, there is a unique consistent distribution $Q(\eta^T, \alpha^T, h^T, \ell^T, o^T)$ capturing the full scenario.

Theorem 9 (Theorem 1 in the paper)**.** *Given a distribution $\{\Pr(\eta_t \mid \eta^{t-1})\}_{t=1}^T$ on high-input strategy functions, a distribution $\{\Pr(\alpha_t \mid \alpha^{t-1})\}_{t=1}^T$ on strategy functions, and a channel $\{\Pr(o_t \mid h^t, \ell^t, o^{t-1})\}_{t=1}^T$, there exists only one consistent measure $Q(\eta^T, \alpha^T, h^T, \ell^T, o^T)$. Furthermore the channel from η^T and α^T to o^T is given by*

$$Q(o_t \mid \eta^t, \alpha^t, o^{t-1}) = \Pr(o_i \mid \eta^t(h^{t-1}, \ell^{t-1}, o^{t-1}), \alpha^t(\ell^{t-1}, o^{t-1}), o^{i-1}). \quad (7)$$

We now show that having probability distributions on deterministic high-input strategy functions and on deterministic strategy functions is sufficient to emulate the probabilistic behavior of the system as given by a probabilistic automaton. The following Theorem 2 shows how given a system we can construct the context that, when combined with that system, produces a consistent Q describing the full scenario.

Theorem 10 (Theorem 2 in the paper)**.** *Let $\mathfrak{I}$ be a probabilistic automaton inducing the joint probability distribution $\Pr(\eta^t, \alpha^t, o^t)$, $1 \leq t \leq T$, on high-input, low-input, and observable-output traces. It is always possible to instantiate the model of in Figure 2 and build a probability distribution $Q(\hat{\eta}^T, \hat{\alpha}^T, \mathcal{H}^T, \mathcal{L}^T, \mathcal{O}^T)$ that corresponds to $\mathfrak{I}$ in the sense that, for every $1 \leq t \leq T$, η^t, α^t, o^t, the equality $Q(\eta^t, \alpha^t, o^t) = \Pr(\eta^t, \alpha^t, o^t)$ holds.*

Moreover, the context that leads to Q is determined as follows.

- *The distribution on $\{\Pr(\eta_t \mid \eta^{t-1})\}_{t=1}^T$ on high-input strategy functions is given by*

$$\Pr(\eta_1) = \Pr(h_1 \mid h^0, \ell^0, o^0) = \Pr(h_1)$$
$$\Pr(\eta_t \mid \eta^{t-1}) = \prod_{hist} \Pr(\hat{\eta}_t \mid \hat{\eta}^{t-1}, \hat{\alpha}^{t-1}, o^{t-1}) \quad 2 \leq t \leq T,$$

  where the index hist of the productory ranges over the full history $h^{t-1}, \ell^{t-1}, o^{t-1}$, and $\hat{\eta}_t = \eta_t(h^{t-1}, \ell^{t-1}, o^{t-1})$, $\hat{\eta}^{t-1} = \eta^{t-1}(h^{t-2}, \ell^{t-2}, o^{t-2})$, and $\hat{\alpha}^{t-1} = \alpha^{t-1}(\ell^{t-2}, o^{t-2})$.

- *The distribution $\{\Pr(\alpha_t \mid \alpha^{t-1})\}_{t=1}^T$ on strategy functions is given by*

$$\Pr(\alpha_1) = \Pr(\ell_1 \mid \ell^0, o^0) = \Pr(\ell_1)$$
$$\Pr(\alpha_t \mid \alpha^{t-1}) = \prod_{pub.hist.} \Pr(\hat{\alpha}_t \mid \hat{\alpha}^{t-1}, o^{t-1}), \quad 2 \leq t \leq T,$$

  where the index $pub.hist.$ of the productory ranges over the public history ℓ^{t-1}, o^{t-1}, and $\hat{\alpha}_t = \alpha_t(\ell^{t-1}, o^{t-1})$ and $\hat{\alpha}^{t-1} = \alpha^{t-1}(\ell^{t-2}, o^{t-2})$.

B. Proofs

We present full proofs of the above theorems. First, here is the proof of Soundness:

Proof: If Q is consistent then $Q(\eta^T, \alpha^T, h^T, \ell^T, o^T)$ is given by

$$\prod_{t=1}^{T} \left[\Pr(\eta_t \mid \eta^{t-1}) \cdot \Pr(\alpha_t \mid \alpha^{t-1}) \cdot \delta_{\{\eta_t(h^{t-1}, \ell^{t-1}, o^{t-1})\}}(h_t) \cdot \right.$$
$$\left. \cdot \delta_{\{\alpha_t(\ell^{t-1}, o^{t-1})\}}(\ell_t) \cdot \Pr(o_t \mid h^t, \ell^t, o^{t-1})\right]. \quad (8)$$

The joint measure for each (η^t, α^t, o^t) can be decomposed as:

$$Q(\eta^t, \alpha^t, o^t) = \sum_{h^t, \ell^t} Q(\eta^t, \alpha^t, h^t, \ell^t, o^t)$$

$= $ (by (8))

$$\sum_{h^t, \ell^t} \prod_{i=1}^{t} \left[\Pr(\eta_i \mid \eta^{i-1}) \cdot \Pr(\alpha_i \mid \alpha^{i-1}) \cdot \right.$$
$$\cdot \delta_{\{\eta_i(h^{i-1}, \ell^{i-1}, o^{i-1})\}}(h_i) \cdot \delta_{\{\alpha_i(\ell^{i-1}, o^{i-1})\}}(\ell_i) \cdot$$
$$\left. \cdot \Pr(o_i \mid h^i, \ell^i, o^{i-1})\right]$$

$= $ (splitting the sum)

$$\Pr(\eta_t \mid \eta^{t-1}) \cdot \Pr(\alpha_t \mid \alpha^{t-1}) \cdot$$
$$\cdot \delta_{\{\eta_t(h^{t-1}, \ell^{t-1}, o^{t-1})\}}(h_t) \cdot \delta_{\{\alpha_t(\ell^{t-1}, o^{t-1})\}}(\ell_t) \cdot$$
$$\cdot \Pr(o_t \mid h^t, \ell^t, o^{t-1}) \cdot$$
$$\cdot \sum_{h^{t-1}, \ell^{t-1}} \prod_{i=1}^{t-1} \left[\Pr(\eta_i \mid \eta^{i-1}) \cdot \Pr(\alpha_i \mid \alpha^{i-1}) \cdot \right.$$
$$\cdot \delta_{\{\eta_i(h^{i-1}, \ell^{i-1}, o^{i-1})\}}(h_i) \cdot \delta_{\{\alpha_i(\ell^{i-1}, o^{i-1})\}}(\ell_i) \cdot$$
$$\left. \cdot \Pr(o_i \mid h^i, \ell^i, o^{i-1})\right]$$

$= $ (by def. of δ and Q)

$$\Pr(\eta_t \mid \eta^{t-1}) \cdot \Pr(\alpha_t \mid \alpha^{t-1}) \cdot$$
$$\cdot \Pr(o_i \mid \eta^t(h^{t-1}, \ell^{t-1}, o^{t-1}), \alpha^t(\ell^{t-1}, o^{t-1}), o^{i-1}) \cdot$$
$$\cdot Q(\eta^{t-1}, \alpha^{t-1}, o^{t-1})$$

$$= \Pr(o_i \mid \eta^t(h^{t-1}, \ell^{t-1}, o^{t-1}), \alpha^t(\ell^{t-1}, o^{t-1}), o^{i-1}) \cdot$$
$$\cdot Q(\eta^t, \alpha^t, o^{t-1}) \quad (9)$$

And from (9) we can conclude:

$$Q(o_t \mid \eta^t, \alpha^t, o^{t-1}) = \Pr(o_t \mid \eta^t(h^{t-1}, \ell^{t-1}, o^{t-1}), \alpha^t(\ell^{t-1}, o^{t-1}), o^{t-1})$$

■

Next, we give the proof of completeness:

Proof: By the laws of probability, $Q(\eta^t, \alpha^t, o^t) = \sum_{h^t, \ell^t} Q(\eta^t, \alpha^t, h^t, \ell^t, o^t)$. We then show that $\sum_{h^t, \ell^t} Q(\eta^t, \alpha^t, h^t, \ell^t, o^t) = \Pr(\eta^t, \alpha^t, o^t)$ by induction on t; *Base case:* $t = 1$. Define $Q(\eta_1 \mid \epsilon, \epsilon, \epsilon) = \Pr(\eta_1(\epsilon))$, $Q(\alpha_1 \mid \epsilon) = \Pr(\alpha_1(\epsilon, \epsilon))$, and $Q(o_1 \mid h_1, \ell_1, \epsilon) = \Pr(o_1 \mid h_1, \ell_1, \epsilon)$. Then:

$$\sum_{\eta^1, \alpha^1} Q(\eta^1, \alpha^1, h^1, \ell^1, o^1) = \sum_{\eta_1, \alpha_1} Q(\eta_1, \alpha_1, h_1, \ell_1, o_1)$$

$= $ (by the chain rule)

$$\sum_{\eta_1, \alpha_1} \left[Q(\eta_1 \mid \epsilon, \epsilon, \epsilon, \epsilon, \epsilon) \cdot Q(\alpha_1 \mid \eta_1, \epsilon, \epsilon, \epsilon, \epsilon) \cdot \right.$$
$$\cdot Q(h_1 \mid \eta_1, \alpha_1, \epsilon, \epsilon, \epsilon) Q(\ell_1 \mid \eta_1, \alpha_1, h_1, \epsilon, \epsilon) \cdot$$

$$
\begin{aligned}
&\cdot Q(o_1 \mid \eta_1, \alpha_1, h_1, \ell_1, \epsilon)] \\
&= \text{(by def. of } Q\text{)} \\
&\quad \sum_{\eta_1,\alpha_1} \big[Q(\eta_1 \mid \epsilon) \cdot Q(\alpha_1 \mid \epsilon) \cdot \delta_{\{\eta_1(\epsilon,\epsilon,\epsilon)\}}(h_1) \cdot \\
&\quad \cdot \delta_{\{\alpha_1(\epsilon,\epsilon)\}}(\ell_1) \Pr(o_1 \mid h^1, \ell^1, \epsilon)\big] \\
&= \text{(by def. of } Q\text{)} \\
&\quad \sum_{\eta_1,\alpha_1} \big[\Pr(\eta_1(\epsilon,\epsilon,\epsilon)) \cdot \Pr(\alpha_1(\epsilon,\epsilon)) \cdot \\
&\quad \cdot \delta_{\{\eta_1(\epsilon,\epsilon,\epsilon)\}}(h_1) \cdot \delta_{\{\alpha_1(\epsilon,\epsilon\}}(\ell_1) \cdot \Pr(o_1 \mid h^1, \ell^1)\big] \\
&= \text{(by def. of } \delta\text{)} \\
&\quad \Pr(h_1)\Pr(\ell_1)\Pr(o_1 \mid h_1, \ell_1) \\
&= \Pr(h_1, \ell_1, o_1) \\
&= \Pr(h^1, \ell^1, o^1)
\end{aligned}
$$

Inductive case: Define $Q(o_t \mid h^t, \ell^t, o^{t-1}) = \Pr(o_t \mid h^t, \ell^t, o^{t-1})$. Define also:

$$
Q(\eta_t \mid \eta^{t-1}) = \prod_{\substack{h^{t-1} \\ \ell^{t-1} \\ o^{t-1}}} \Pr(\hat{\eta}_t \mid \hat{\eta}^{t-1}, \hat{\alpha}^{t-1}, o^{t-1})
$$

$$
Q(\alpha_t \mid \alpha^{t-1}) = \prod_{\substack{\ell^{t-1} \\ o^{t-1}}} \Pr(\hat{\alpha}_t \mid \hat{\alpha}^{t-1}, o^{t-1})
$$

where $\hat{\eta}_t = \eta_t(h^{t-1}, \ell^{t-1}, o^{t-1})$, $\hat{\eta}^{t-1} = \eta^{t-1}(h^{t-2}, \ell^{t-2}, o^{t-2})$, $\hat{\alpha}_t = \alpha_t(\ell^{t-1}, o^{t-1})$, and $\hat{\alpha}^{t-1} = \alpha^{t-1}(\ell^{t-2}, o^{t-2})$.

Then we can derive:

$$
\begin{aligned}
&\sum_{\eta^t,\alpha^t} Q(\eta^t, \alpha^t, h^t, \ell^t, o^t) \\
&= \text{(by the chain rule)} \\
&\quad \sum_{\eta^t,\alpha^t} \big[Q(\eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \\
&\quad \cdot Q(\eta_t \mid \eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \\
&\quad \cdot Q(\alpha_t \mid \eta^t, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \\
&\quad \cdot Q(h_t \mid \eta^t, \alpha^t, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \\
&\quad \cdot Q(\ell_t \mid \eta^t, \alpha^t, h^t, \ell^{t-1}, o^{t-1}) \cdot \\
&\quad \cdot Q(o_t \mid \eta^t, \alpha^t, h^t, \ell^t, o^{t-1})\big] \\
&= \text{(Because } Q \text{ is consistent)} \\
&\quad \sum_{\eta^t,\alpha^t} \big[Q(\eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot Q(\eta_t \mid \eta^{t-1}) \cdot \\
&\quad \cdot Q(\alpha_t \mid \alpha^{t-1}) \cdot \delta_{\{\eta_t(h^{t-1},\ell^{t-1},o^{t-1})\}}(h_t) \cdot \\
&\quad \cdot \delta_{\{\alpha_t(\ell^{t-1},o^{t-1})\}}(\ell_t) \cdot Q(o_t \mid h^t, \ell^t, o^{t-1})\big] \\
&= \text{(By construction of } Q\text{)} \\
&\quad \sum_{\eta^t,\alpha^t} \big[Q(\eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \\
&\quad \cdot \left(\prod_{\substack{h'^{t-1} \\ \ell'^{t-1} \\ o'^{t-1}}} \Pr(\hat{\eta'}_t \mid \hat{\eta'}^{t-1}, \hat{\alpha'}^{t-1}, o'^{t-1}) \right) \cdot \\
&\quad \cdot \left(\prod_{\substack{\ell'^{t-1} \\ o'^{t-1}}} \Pr(\hat{\alpha'}_t \mid \hat{\alpha'}^{t-1}, o'^{t-1}) \right) \cdot \\
&\quad \cdot \delta_{\{\eta_t(h^{t-1},\ell^{t-1},o^{t-1})\}}(h_t) \cdot \delta_{\{\alpha_t(\ell^{t-1},o^{t-1})\}}(\ell_t) \cdot \\
&\quad \cdot \Pr(o_t \mid h^t, \ell^t, o^{t-1})\big] \qquad (10)
\end{aligned}
$$

where $\hat{\eta'}_t = \eta_t(h'^{t-1}, \ell'^{t-1}, o'^{t-1})$, $\hat{\eta'}^{t-1} = \eta^{t-1}(h'^{t-2}, \ell'^{t-2}, o'^{t-2})$, $\hat{\alpha'}_t = \alpha_t(\ell'^{t-1}, o'^{t-1})$, and $\hat{\alpha'}^{t-1} = \alpha^{t-1}(\ell'^{t-2}, o'^{t-2})$.

We can use the definition of δ to derive from (10):

$$
\begin{aligned}
&\sum_{\eta^t,\alpha^t} Q(\eta^t, \alpha^t, h^t, \ell^t, o^t) \\
&= \sum_{\substack{\eta^t,\alpha^t \\ \eta_t(h^{t-1},\ell^{t-1},o^{t-1})=h_t \\ \alpha_t(\ell^{t-1},o^{t-1})=\ell_t}} \big[Q(\eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \\
&\quad \cdot \left(\prod_{\substack{h'^{t-1} \\ \ell'^{t-1} \\ o'^{t-1}}} \Pr(\hat{\eta'}_t \mid \hat{\eta'}^{t-1}, \hat{\alpha'}^{t-1}, o'^{t-1}) \right) \cdot \\
&\quad \cdot \left(\prod_{\substack{\ell'^{t-1} \\ o'^{t-1}}} \Pr(\hat{\alpha'}_t \mid \hat{\alpha'}^{t-1}, o'^{t-1}) \right) \cdot \Pr(o_t \mid h^t, \ell^t, o^{t-1})\Bigg] \\
&= \text{(By splitting the sum)} \\
&\quad \sum_{\substack{\eta^{t-1} \\ \alpha^{t-1}}} Q(\eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \Pr(o_t \mid h^t, \ell^t, o^{t-1}) \cdot \\
&\quad \sum_{\substack{\eta_t,\alpha_t \\ \eta_t(h^{t-1},\ell^{t-1},o^{t-1})=h_t \\ \alpha_t(\ell^{t-1},o^{t-1})=\ell_t}} \left[\left(\prod_{\substack{h'^{t-1} \\ \ell'^{t-1} \\ o'^{t-1}}} \Pr(\hat{\eta'}_t \mid \hat{\eta'}^{t-1}, \hat{\alpha'}^{t-1}, o'^{t-1}) \right) \cdot \right. \\
&\quad \left. \cdot \left(\prod_{\substack{\ell'^{t-1} \\ o'^{t-1}}} \Pr(\hat{\alpha'}_t \mid \hat{\alpha'}^{t-1}, o'^{t-1}) \right) \right] \\
&= \text{(By distributivity)} \\
&\quad \sum_{\substack{\eta^{t-1} \\ \alpha^{t-1}}} Q(\eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \Pr(o_t \mid h^t, \ell^t, o^{t-1}) \cdot \\
&\quad \left[\sum_{\substack{\eta_t \\ \eta_t(h^{t-1},\ell^{t-1},o^{t-1})=h_t}} \left(\prod_{\substack{h'^{t-1} \\ \ell'^{t-1} \\ o'^{t-1}}} \Pr(\hat{\eta'}_t \mid \hat{\eta'}^{t-1}, \hat{\alpha'}^{t-1}, o'^{t-1}) \right) \right] \cdot \\
&\quad \cdot \left[\sum_{\substack{\alpha_t \\ \alpha_t(\ell^{t-1},o^{t-1})=\ell_t}} \cdot \left(\prod_{\substack{\ell'^{t-1} \\ o'^{t-1}}} \Pr(\hat{\alpha'}_t \mid \hat{\alpha'}^{t-1}, o'^{t-1}) \right) \right] \\
&= \text{(By applying Lemma 2 twice)}
\end{aligned}
$$

$$\sum_{\substack{\eta^{t-1}\\ \alpha^{t-1}}} \left[Q(\eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \Pr(o_t \mid h^t, \ell^t, o^{t-1}) \cdot \Pr(h_t \mid h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \Pr(\ell_t \mid \ell^{t-1}, o^{t-1})\right]$$

$$= \text{(By reorganizing the sum)}\ \Pr(o_t \mid h^t, \ell^t, o^{t-1}) \cdot \Pr(\ell_t \mid \ell^{t-1}, o^{t-1}) \cdot \Pr(h_t \mid h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \sum_{\substack{\eta^{t-1}\\ \alpha^{t-1}}} Q(\eta^{t-1}, \alpha^{t-1}, h^{t-1}, \ell^{t-1}, o^{t-1})$$

$$= \text{(By induction hypothesis)}\ \Pr(o_t \mid h^t, \ell^t, o^{t-1}) \cdot \Pr(\ell_t \mid \ell^{t-1}, o^{t-1}) \cdot \Pr(h_t \mid h^{t-1}, \ell^{t-1}, o^{t-1}) \cdot \Pr(h^{t-1}, \ell^{t-1}, o^{t-1})$$

$$= \text{(By the chain rule)}\ \Pr(h^t, \ell^t, o^t)$$

■

Lemma 2. *Let $\mathcal{X}, \mathcal{Y}$ be non-empty finite sets, and let $\tilde{x} \in \mathcal{X}, \tilde{y} \in \mathcal{Y}$. Let $q : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ be a function such that, for every $x \in \mathcal{X}$, we have: $\sum_{y \in \mathcal{Y}} q(x, y) = 1$. Then:*

$$\sum_{\substack{f \in \mathcal{X} \to \mathcal{Y}\\ f(\tilde{x}) = \tilde{y}}} \prod_{x \in \mathcal{X}} q(x, f(x)) = q(\tilde{x}, \tilde{y})$$

B. Memory Limited Adversaries

Our general information metric from Section IV-C is based on an adversary that can optimally consider past history. In addition, for any natural number M we can also consider optimal, albeit *memory-limited* adversaries whose strategy function takes in only the most recent M observations and low inputs. Let actf^M be the types for such strategies.

Definition 11. For a memory limit M, the *(memory limited) dynamic gain* in a `model` is the gain an optimal memory adversary is expected to achieve under a `gain_func`. We will note this quantity as below:

$$\mathbb{D}^M_{\texttt{gain\_func}}(\texttt{model}) \stackrel{\text{def}}{=} \max_{s \in \mathit{actf}^M} \mathbb{E}\left[X_{\texttt{model gain\_func s}}\right] = \mathbb{E}\left[X_{\texttt{model gain\_func opt\_strat\_M}}\right]$$

1) Optimal (memory limited) adversary: We will refer to `opt_strat_M`:actf^M as optimal memory limited strategies. We can construct such as strategy by using a slight modification of the procedure give in Section IV-C.

Let `fM = take M` be a function that returns the length M prefix of a list (and *forgets* the rest).For every `n, lows, obss` in decreasing order of `n` and length of history, the decisions of such an optimal, though memory limited adversary, `opt_stratM`, are constructed using the following modified definition:

$$G[\texttt{n, fM lows, fM obss}] \stackrel{\text{def}}{=} \max\{ \max_{\texttt{e}\in\mathcal{E}} \mathbb{E}[X_{\texttt{gain}} \mid X_{\texttt{fM hist.lows}} = \texttt{fM lows}, X_{\texttt{fM hist.obss}} = \texttt{fM obss}, X_{\texttt{hist.atk}} = \texttt{Some e}], \max_{\texttt{l}\in\mathcal{L}} \mathop{\mathbb{E}}_{\texttt{o}\leftarrow O^{n+}_{\texttt{lows,obss}}(l)} [G[\texttt{n + 1}, \texttt{fM (lows @ [l])}, \texttt{fM (obss @ [o])}]]\}$$

The definition of $O^{n+}_{\texttt{lows,obss}}(l)$ needs to likewise be modified to forget.

$$\Pr\left(O^{n+}_{\texttt{lows,obss}}(l) = \texttt{o}\right) \stackrel{\text{def}}{=} \Pr(X_{\texttt{hist.obss.(n+1)}} = \texttt{o} \mid X_{\texttt{fM hist.lows}} =_{(n+1)} \texttt{fM (lows @ [l])}, X_{\texttt{fM hist.obss}} =_n \texttt{fM obss})$$

Memory-limited expected gain, or $\mathbb{D}^M_{\texttt{gain\_func}}(\texttt{model})$ is then equal to $G[0, [], []]$. By replacing optimal gain $\mathbb{D}_\cdot(\cdot)$ with $\mathbb{D}^M_\cdot(\cdot)$ in the metrics given in Section IV-D we can construct memory-limited versions of them.

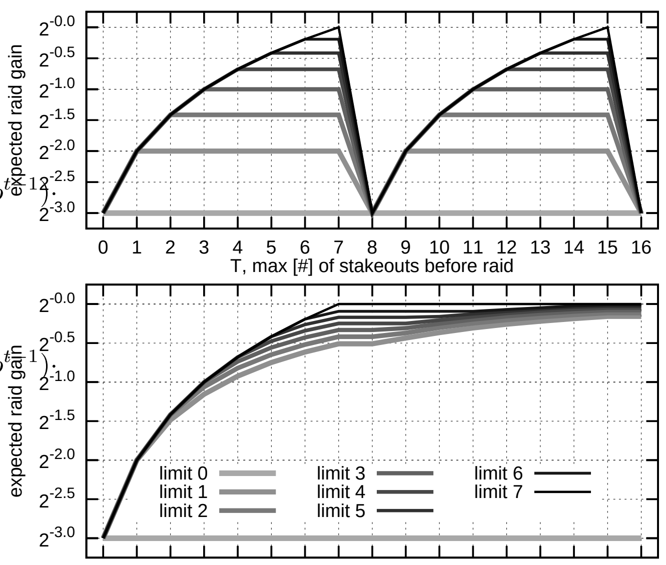


Fig. 12. Expected raid gain over time with changing stash `move_stash 8` and limited memory given stakeouts `stakeout_rotate` with (top) non-adaptive `raid_non_adapt` and (bottom) wait-adaptive raids `raid_wait_adapt`.

2) How does an adversary memory limit impact dynamic gain?: To see the effect of memory limitation, we construct a variant of stash/raid experiment from Section VI. In this version the stash will change location every 8 time steps. For whatever reason, the police have trouble keeping up with paperwork and forget the results of any stakeout older than m time units in the past. If they do not perform raids adaptively this severely limits their chances of success.

In Figure 12(top) we see the periodic gain of limited adversaries without ability to wait. An adversary with memory limit m does increasingly better until they observe m stakeouts at which point their expected gain becomes constant until the change function applies every 8 time steps at which point their gain resets. This results in the higher-memory adversaries diverging from lower-memory adversaries with change leveling the playing field periodically.

On the other hand, even an adversary with a very limited memory (remembering only the most recent observation) is able to do well if they can wait until the right moment. For wait-adaptive adversaries the periodic diverging behavior also occurs to some extent but is overshadowed by the monotonic trend (see Figure 12(bottom)). Even with limited memory, the most significant part of an adversary's behavior is to wait until they see a single successful stakeout.

C. Expressing Existing Metrics

In this section we present the details of how our model expresses existing information flow metrics as summarized in Section IV-D. In these metrics the adversary has no choices to make, only attacks at a fixed point after a fixed number of observations, with observations that only depend on a static high value.

Lemma 1. *If* `model` *and* `gain_func` *are static then dynamic gain simplifies to the following.*

$$\mathbb{D}_{\texttt{gain\_func}}(\texttt{model}) = \underset{\texttt{obss} \leftarrow X_{\texttt{hist.obss}}}{\mathbb{E}} \max_{e} \mathbb{E}(X_{\texttt{gain}} \mid X_{\texttt{hist.obss}} = \texttt{obss}, X_{\texttt{hist.atk}} = \texttt{Some e})$$

Proof: Under the static restrictions, we can rewrite Equations 4 and 5. We will omit the `lows` parts of the definitions and of the map G below as there are no low adversary choices.

$$G[\texttt{obss}] \overset{\text{def}}{=}$$
$$\text{when } \texttt{n = length obss = T}: \max_{\texttt{e} \in \mathcal{E}} \mathbb{E}[X_{\texttt{gain}} \mid X_{\texttt{hist.obss}} = \texttt{obss}, X_{\texttt{hist.atk}} = \texttt{Some e}]$$
$$\text{when } \texttt{n < tmax}: \underset{\texttt{o} \leftarrow O^{n+}_{\texttt{obss}}}{\mathbb{E}} [G[\texttt{obss @ [o]}]]$$

Here we are not considering the adversary attacking before `T` hence the equation can be split into two parts; one in which they are only considering attacking, and one in which they are going to observe. Since they have no choices of low inputs, the `T` in the latter part disappeared from the equation as well. Likewise, the definition of the r.v. representing the next observation does not depend on low and is simplified:

$$\Pr\left(O^{n+}_{\texttt{obss}} = o\right) \overset{\text{def}}{=} \Pr(X_{\texttt{hist.obss.(n)}} = o \mid X_{\texttt{hist.obss}} =_n \texttt{obss})$$

We can now unroll the definition of G in Equation 4, giving us the following:

$$G[[]] = \underset{\texttt{o}_1 \leftarrow O^{0+}_{\texttt{[]}}}{\mathbb{E}} \underset{\texttt{o}_2 \leftarrow O^{1+}_{\texttt{[o}_1\texttt{]}}}{\mathbb{E}} \cdots \underset{\texttt{o}_T \leftarrow O^{(T-1)+}_{\texttt{[o}_1\texttt{; o}_2\texttt{; ...; o}_{T-1}\texttt{]}}}{\mathbb{E}} \max_{\texttt{e} \in \mathcal{E}} \mathbb{E}(X_{\texttt{gain}} \mid X_{\texttt{hist.obss}} = \texttt{[o}_1\texttt{; ...; o}_T\texttt{]}, X_{\texttt{hist.atk}} = \texttt{Some e})$$

Replacing the sequence of T observations in the above with one list `obss` gives us the claim of the lemma. ■

Lemma 1 will make it clear in the following sections how we model existing information flow metrics. All of these metrics are expressed using the same pattern seen in the equation, only varying in their gain functions. We will use `secret` to express `last hist.highs`, the sole unchanging secret value for brevity.

Min-vulnerability: The notion of *min-vulnerability* corresponds to an equality gain function.

```
let gain_min_vul: gainf =
  fun (hist: history) (exp: ℰ) ->
    if secret == exp then 1.0 else 0.0
```

Theorem 4. *In a static* `model`*, the min-vulnerability of the secret conditioned on the observations is equivalent to dynamic gain using the* `gain_min_vul` *gain function.*

$$\mathbb{D}_{\texttt{gain\_min\_vul}}(\texttt{model}) = \mathbb{V}(X_{\texttt{secret}} \mid X_{\texttt{hist.obss}})$$

Proof: The goal of the attacker assumed in min-vulnerability is evident from this function; they are directly guessing the secret, and they only have one chance to do it.

$$\mathbb{V}(X) \overset{\text{def}}{=} \max_{x} \Pr(X = x)$$
$$\mathbb{V}(X \mid Y) \overset{\text{def}}{=} \underset{y \leftarrow Y}{\mathbb{E}} [\mathbb{V}(X \mid Y = y)]$$

The first definition is prior min-vulnerability whereas the second is the posterior. Posterior vulnerability is the expectation of vulnerability of X given that an adversary observes Y.

Using `gain_min_vul`, dynamic gain according to Lemma 1 simplifies to the following, noting that $\mathbb{E}(X_{\texttt{if secret = exp then 1.0 else 0.0}}) = \Pr(X_{\texttt{secret}} = \texttt{exp}) \cdot 1.0 + \Pr(X_{\texttt{secret}} \neq \texttt{exp}) \cdot 0.0 = \Pr(X_{\texttt{secret}} = \texttt{exp})$:

$$\mathbb{D}_{\texttt{gain\_min\_vul}}(\texttt{model}) = \underset{\texttt{obss} \leftarrow X_{\texttt{hist.obss}}}{\mathbb{E}} \max_{\texttt{e} \in \mathcal{E}} \Pr(X_{\texttt{secret}} = \texttt{e} \mid X_{\texttt{hist.obss}} = \texttt{obss})$$

The dynamic gain above is identical to posterior min-vulnerability of $X_{\texttt{secret}}$ given $X_{\texttt{hist.obss}}$. Note that we assumed the set of exploits is the same as the set of secrets in the restricted model. ■

Generalized gain functions: g-vulnerability based on generalized gain functions can also be expressed in terms of the restricted model.Let `g` be a generalized gain function, returning a `float` between 0.0 and 1.0, then we have:

```
let gain_gen_gain (g: ℋ → ℰ → float): gainf =
  fun (hist: history)
      (exp: ℰ) : float ->
  g secret exp
```

Theorem 5. *In a static* `model` *the g-vulnerability of* `g` *is equivalent to dynamic gain using* `gain_gen_gain g` *gain function.*

$$\mathbb{D}_{\texttt{gain\_gen\_gain}}(\texttt{model}) = \mathbb{V}_g(X_h \mid X_{\texttt{hist.obss}})$$

Proof: Equation in Lemma 1 (replacing `gain` with `g h a`) directly corresponds with posterior g-vulnerability of [14]. ∎

Guessing-entropy: Guessing entropy, characterizing the expected number of guesses an optimal adversary will need in order to guess the secret. We let attacks be lists of highs (really permutations of all highs). The attack permutation corresponds to an order in which secrets are to be guessed. We then define expected gain to be proportional to how early in that list of guesses is.

```
type ℰ = ℋ list

let pos_of (h: ℋ) (exp: ℋ list) =
  (* compute the position of h in exp *)

let gain_guess_ent: gainf =
  fun (hist: history)
      (exp: ℰ) =
  -1.0 * (1.0 + (position_of secret exp))
```

Note that we negate the gain as an adversary would optimize for the minimum number of guesses, not the maximum. Guessing entropy, written $\mathbb{G}(X)$ is the negation of dynamic gain:

Theorem 6. *In a static* `model`*, guessing entropy is equivalent to (the negation of) dynamic gain using the* `gain_guess_ent` *gain function.*

$$-\mathbb{D}_{\texttt{gain\_guess\_ent}}(\texttt{model}) = \mathbb{G}(X_{\texttt{secret}} \mid X_{\texttt{hist.obss}})$$

Proof: The definition of guessing entropy encapsulates an adversary attacking a system in a particular manner:

$$\mathbb{G}(X) \overset{\text{def}}{=} \sum_{i=1}^{N} i \cdot \Pr(X = x_i)$$

The N values x_i above are assumed ordered with decreasing probability. The optimal adversary knowing a secret distributed according to r.v. X is trying to guess it with the fewest expected guesses. This adversary would thus guess in decreasing order of probability.

We show that this order achieves the maximum gain for the `gain_guess_ent` gain function. Let us fix some observation list `obss` and focus on the random variable $X_{\texttt{gain}}$ conditioned on $X_{\texttt{hist.obss}} = \texttt{obss}, X_{\texttt{hist.atk}} = \texttt{Some perm}$. The permutation `perm = [x₁; ...; x_N]` is an ordering of $\mathcal{H}$ in decreasing probability according to $\Pr(X_{\texttt{secret}} = \texttt{x}_i \mid X_{\texttt{hist.obss}} = \texttt{obss})$. Starting from Lemma 1, the expected gain for this attack, given the fixed observations is:

$$\begin{aligned}
&\mathbb{E}(X_{\texttt{gain}} \mid X_{\texttt{hist.obss}} = \texttt{obss}, \\
&\qquad X_{\texttt{hist.atk}} = \texttt{Some perm}) = \\
&\sum_{i=-N}^{-1} i \cdot \Pr(X_{\texttt{-1.0*(1.0+(pos\_of secret perm))}} = i \mid \\
&\qquad X_{\texttt{hist.obss}} = \texttt{obss}) = \\
&-\sum_{i=1}^{N} i \cdot \Pr(X_{\texttt{pos\_of secret perm}} = i - 1 \mid \\
&\qquad X_{\texttt{hist.obss}} = \texttt{obss}) = \\
&-\sum_{i=1}^{N} i \cdot \Pr(X_{\texttt{secret}} = \texttt{x}_i \mid X_{\texttt{hist.obss}} = \texttt{obss})
\end{aligned}$$

No other ordering of the high values can produce an expected gain larger than this. Consider an ordering that does not have the highest probably secret x_1 guessed first. That is, `perm'` $= [x_i; ...; x_1; ...]$ where x_1 is in position $j > 1$. Let $d \overset{\text{def}}{=} \Pr(X_{\texttt{secret}} = x_1 \mid ...) - \Pr(X_{\texttt{secret}} = x_i \mid ...) \geq 0$. Swapping x_i and x_1 in this permutation will change the expected gain for this order by $j \cdot d - 1 \cdot d = d \cdot (j - 1) \leq 0$. Thus the adversary could do at least as well or better by guessing the most probable secret first. The same argument can be made for the second most probable secret (for second guess) and so on. Any permutation that is not `perm` can be reordered using this process into `perm` while maintaining or improving expected gain. Thus `perm` must achieve the maximal expected gain. ∎