From Operational Models to Information Theory; Side Channels in pGCL with Isabelle

David Cock

NICTA and University of New South Wales
David.Cock@nicta.com.au

Abstract. In this paper, we formally derive the probabilistic security predicate (expectation) for a guessing attack against a system with side-channel leakage, modelled in pGCL. Our principal theoretical contribution is to link the process-oriented view, where attacker and system execute particular model programs, and the information-theoretic view, where the attacker solves an optimal-decoding problem, viewing the system as a noisy channel. Our practical contribution is to illustrate the selection of probabilistic loop invariants to verify such security properties, and the demonstration of a mechanical proof linking traditionally distinct domains.

1 Introduction

This paper presents a formally-verified proof of the optimality of a Bayesian attack against an interactive system with side-channel leakage. This is an unsurprising result from a security and machine-learning perspective, and would typically just be assumed. Formalising the proof, however, requires effort. The point, of course, is that the proof is straightforward exactly where it should be, in the mechanical plugging-together of pieces, but interesting (or difficult), only where it must be, namely in choosing a model that faithfully represents our intuitive understanding of the problem, and in the 'creative moment' of choosing an appropriate loop invariant. The result itself will be unsurprising to anybody familiar with information theory or machine learning, but is the critical step in linking this typically mathematical field with the more pragmatic world of concrete operational semantics. As we will briefly detail in the final section of this paper, and as we have presented elsewhere[Cock, 2014], we can then apply the tools of one domain to the other, to tease out deeper theoretical results.

The contribution of this paper is a worked mechanical proof that conducting an optimal guessing attack, in the presence of side-channel leakage, reduces to an optimal decision problem on a corresponding noisy channel, expressed in purely probabilistic terms, with all artefacts of the operational model eliminated. This model, in turn, is carefully selected to make the connection to the intuitive security property as clear as possible, but also flexible enough to incorporate realistic-scale systems in the manner demonstrated in our previous work[Cock, 2013]. To reiterate, a straightforward proof of such a cross-cutting result, requiring deep

thought only at those points where it really should, demonstrates that the infrastructure of mechanisation is mature enough to allow us to transfer a security result from a detailed, fully-implemented system, through a probabilistic operational semantics[Cock, 2012], to a fully probabilistic or information-theoretic setting.

1.1 Side Channels and Guessing Attacks

The context of this work is our ongoing project[Cock, 2014] to demonstrate that we can transfer probabilistic information-flow properties from an abstract information-theoretic model, down to a concrete verified implementation, leveraging an existing correctness proof (for example that of the seL4 microkernel[Klein et al., 2009]). Previously[Cock, 2013], we demonstrated that the top-level result of that refinement proof could be embedded in the probabilistic language pGCL[McIver and Morgan, 2004], and lemmas about the abstract specification lifted into a probabilistic context. We now demonstrate that we can take one further step, and lift a probabilistic security property away from implementation details entirely, and reason solely at an abstract level.

The reason that this approach works, and that we are able to claim that it is applied to a real system is precisely the embedding result just mentioned. We were able to take the top-level specification of seL4, and embed it into pGCL in such a way that annotations of the original specification are preserved, and the refinement orders correspond. Therefore, any refinement-sound result on the probabilistic specification (such as that shown in this paper), is preserved by the full seL4 refinement stack and therefore applies to the low-level kernel implementation.

This security property is derived from our *threat model*: A guessing attack, against a system with a side channel. We define a side channel to be any information provided by the system to a client that a) depends on a *secret*, and b) is not allowed by the system's specification.

As a motivating example, consider an authentication system in which the client supplies some shared secret (for example a password) to the system in order to prove its identity. The system must compare the supplied secret to its stored version, and then either grant or deny access. Such a system has some degree of unavoidable leakage: a malicious client can guess secrets one-by-one and (assuming the set is finite), must eventually get the right one. In practice, we hope that the space of secrets is sufficiently large that such an attack is impractical. More interesting is whether the system leaks anything *beyond* this unavoidable amount, and to what extent any such additional *side-channel* leakage increases the vulnerability of the system (or the attacker's chance of success).

This leakage might be in the form of variable runtime, power consumption, radiation, or any other property not covered by the system's specification. We assume that continuous channel values are measured with some finite precision by the attacker, and are thus effectively quantised into some finite set of *outputs*. A common property of such side-channel outputs is that they are *stochastic*: there

is no functional relation between the secret and the observed side-channel value. There may, however, exist some *probabilistic* relationship.

If there were a functional relation between the secret, $S \in \mathbb{S}$, and observation $o \in \mathbb{O}$, i.e. we are able to find some $F :: \mathbb{S} \Rightarrow \mathbb{O} \Rightarrow \mathit{bool}$, such that[1]

$$\forall S.\ \exists! o.\ F\ S\ o$$

then we could proceed to reason about whether any information about S is leaked by revealing o. In particular, if F is invariant under changing S, or $\forall S\, S'\, o.\ F\ S\ o = F\ S'\ o$, then we can say with confidence that o gives no information about S.

In the probabilistic case, in general, no such relation exists and we must be satisfied with a probabilistic relationship, the *conditional probability* $P :: \mathbb{S} \Rightarrow \mathbb{O} \Rightarrow [0,1]$, where:

$$\forall S.\ \sum_{o} P\ S\ o = 1$$

Here, we interpret $P\ S\ o$ as the probability of producing observation o, assuming that the secret is S, for which we write $P[o|S]$, or 'the probability of o given S'.

Note that exactly the same invariance reasoning applies: if $\forall S\, S'\, o.\ P\ S\ o = P\ S'\ o$ i.e. the distribution of outcomes does not depend on S, then observing o tells the attacker nothing about S. Note also that if we define the boolean embedding operator $\langle\!\langle \cdot \rangle\!\rangle :: (\alpha \Rightarrow \mathit{bool}) \Rightarrow \alpha \Rightarrow [0,1]$ by:

$$\langle\!\langle Q \rangle\!\rangle S = \mathsf{if}\ Q\ S\ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

then we can view a functional relation $(\lambda S\, o.\ F\ S\ o)$ as the special case of a probabilistic relation $(\lambda S\, o.\ \langle\!\langle F\ S \rangle\!\rangle\ o)$ which, for each S, assigns 100% probability to exactly one o.

Given a non-injective relation, (or overlapping distributions), the attacker will generally not be able to determine the secret with certainty: As long as at least two secrets can produce the observed output with non-zero probability, then both are possible. In the probabilistic case however (and in contrast to the functional case), the attacker may be able to distinguish possible secrets by *likelihood*. All other things being equal, if secret S produces the observed output o with *greater conditional probability* than S' i.e. $P[o|S] > P[o|S']$, then S is a more likely candidate than S'.

Intuitively, if the attacker wants to *use* the secret that it has guessed, say to attempt to authenticate, then it should use the one that it considers to be *most likely to be correct*, in order to maximise its probability of success. The attacker thus needs to order possible secrets by $P[S|o]$, or the probability that the secret is S, having observed o. In order to calculate this, however, the attacker needs to know $P[S]$, or the prior probability of a given secret: even if $P[o|S] > P[o|S']$, and so S is more likely that S' on the basis of the evidence, if the secret is overwhelmingly more likely to be S' than S in the first case (perhaps S' is the

[1] The Isabelle/HOL syntax $\exists! x.\ P\ x$ reads 'there exists a *unique* x satisfying P'

password '12345'), then even given the evidence it may still be more likely than S, albeit less so than initially.

The calculation of this inverse conditional probability $P[S|o]$, from the forward probability $P[o|S]$, is made via Bayes' rule:

$$P[S|o] = \frac{P[o,S]}{P[o]}$$

This states that the probability of the hypothesis (that the secret is S) given the evidence (that o was observed), is the joint probability of the hypothesis and the evidence ($P[o,S] = P[o|S]P[S]$), divided by the probability of the evidence ($P[o]$, or the likelihood of observing o across all secrets).

Thus, as $P[o]$ does not depend on S, the most likely secret (given that we have observed o), is that which maximises $P[o,S]$. Selecting S so as to maximise $P[o,S]$ in this way is known as the *maximum a posteriori* (MAP) rule: selecting the hypothesis with the highest a posteriori (after the fact) probability. In information theory, this is the optimal solution to the *decoding problem*: If the conditional probability $P[o|S]$ summarises the *channel matrix* of some noisy channel, i.e. $P[o|S]$ is the probability that symbol o is received, when S is transmitted, the decoding problem is that faced by the receiver: to choose the most likely transmitted symbol with which to decode each received symbol. This is exactly analogous to the challenge facing an attacker attempting to guess the secret given a side-channel observation, and both are solved optimally if the MAP rule is observed.

There is one way, however, in which the attacker is not like the receiver on a noisy channel: where the receiver is *passively* listening to the output, and making a best-effort reconstruction of the input, the attacker is *actively* probing the system, and has a second source of information in knowing whether it manages to authenticate, or is refused. The additional leakage implies that the most-likely secret (on the current evidence) is not necessarily the right one to guess. In particular, the attacker should never guess the same thing twice: if it's been refused once, it'll be refused again.

The criterion for success is also slightly different. In the noisy channel example, the decoder is successful if it chooses the correct S, and fails otherwise: there are no second chances. In the authentication example, however, if the attacker fails on its first guess (as it will most likely do), it can simply try again. Given a finite space of secrets therefore, an attacker that does not repeat guesses is guaranteed to eventually guess correctly. The trick is, of course, that in a large enough space, the chance of doing so in any practical timeframe is essentially zero. The security measure is therefore not 'the probability that the attacker guesses correctly' (which is 100%, eventually), but 'the probability that the attacker guesses correctly *in no more than n tries*'. The parameter n can then be set to an appropriate value for the system in question. For example, a login service may allow only n attempts before blacklisting a particular client, in which case we are asking for the probability that the system is compromised before the countermeasure (blacklisting) kicks in.

1.2 What the Proof Shows

The proof in Section 2 shows that what we expect, based on the informal reasoning above is, in fact, exactly what we get. As previously described, the bulk of the proof is relatively mechanical but, thanks to mechanisation, rather short. The interesting points, on which we will focus, are, again as already mentioned: in establishing the faithfulness of the formal model to the above intuitive description, and in the choice of loop invariant, given which the rest of the result follows straightforwardly.

The last thing we must mention before diving in is our two major assumptions, which are *implicit* in the above discussion, and in our choice of model, but need to be stated explicitly:

- We assume that multiple observations are independent, given the secret. The underlying assumption is that there is no lasting effect between two invocations: the result of one probe has no influence on the result of the next:
$$P(o, o'|S) = P(o|S)P(o'|S) \tag{1}$$
This assumption could be lifted, for example by moving to an n-Markov model, which should be expressible in the same framework, if this turns out to be necessary in modelling a real system. For simplicity however, we start with the '0-Markov', or independent case.
- We assume that the side-channel output (o) depends only on the secret (S), and not on the attacker-supplied input.
This assumption is more important, and more limiting, than the first. It is easy to construct systems where side-channel output depends on attacker-supplied input, and in making this assumption, we are excluding them from consideration. Such fully-dynamic attacks are the subject of current research, and there is as yet no elegant information-theoretic interpretation. We must thus be content for now with our partially-dynamic attack, where the attacker's actions may be dynamically determined, but the conditional probabilities are not.

1.3 An Overview of pGCL

The probabilistic imperative language pGCL is a descendent of GCL[Dijkstra, 1975], that incorporates both classical nondeterminism and probability. Common primitives are provided, including:

- Sequential composition: $a\,;;\,b$, or "do a then b".
- Variable update: $x{:=}(\lambda s.\ e\ s)$, or "update variable x, with the value of expression e in the current state, s".
- Looping: *do* $G \longrightarrow$ *body od*, or "execute *body* for as long as G (the guard) holds".

We may also make both classically nondeterministic and probabilistic choices, which we present here in the form of choices over values:

- Unconstrained (demonic) choice: *any* x, or "assign any value to variable x".
- Constrained (probabilistic) choice: *any* x *at* $(\lambda s\ v.\ P\ s\ v)$, or "choose value v, with probability P (which may depend on the state), and assign it to variable x".
- Constrained name binding (let expressions): *bind* n at $(\lambda s\ v.\ P\ s\ v)$ *in* $a\ n$. This is almost equivalent to the previous operation, but instead of modifying the state, simply binds the name n, which parameterises the program a. As indicated, this is simply a probabilistic let expression.

Expectations In pGCL, we are no longer constrained to reasoning about boolean predicates on the state (pre- and post-conditions, for example), but can now consider any bounded, non-negative real-valued function (an *expectation*). We can annotate program fragments just as in GCL:

$$Q \Vdash \mathrm{wp}\ a\ R$$

Traditionally, we would read this predicate-entailment relation as a Hoare triple: if Q holds initially, then R will hold after executing a. The literal meaning being that Q implies the *weakest precondition* of R. In pGCL, both Q and R are real-valued, and implication is replaced by pointwise comparison. The weakest precondition is replaced by the weakest pre-*expectation*, such that the equation now reads: Q is everywhere lower than the weakest preexpectation of R.

The action of programs is defined such that the weakest preexpectation of R is the expected value of R after executing a, or the sum over all possible states s of $R\ s$, weighted by the probability of ending in state s. If R is the probability function associated with some predicate i.e. R is 1 whenever the predicate holds, and 0 otherwise, then the weakest preexpectation is now the *probability*, calculated in the initial state, that the predicate holds in the final state (the expected value of a probability is itself a probability). In this case, the above annotation states that the probability that the predicate holds finally, if we begin in state s is *at least* $Q\ s$. The embedding, in the above fashion, of a predicate S as an expectation is written $«S»$.

Note that if $Q\ s$ is 1, this reduces to the classical case: If Qs is true (has probability 1), then Rs will hold (with probability at least[2] 1.).

Invariants A loop *invariant* in pGCL is an expectation that is preserved by every iteration of the loop:

$$I * «G» \Vdash \mathrm{wp}\ \mathit{body}\ I$$

In the standard (non-probabilistic) setting, this states that if the invariant I and guard G hold before the body executes, the invariant still holds afterwards. In our probabilistic setting, we instead have that (as G is $\{0,1\}$-valued), the probability

[2] The healthiness conditions for the underlying semantics ensures that the probability is in fact exactly one.

of establishing I from any state where the loop body executes (satisfying G), is at least I, or that I is a *fixed point* of the guarded loop body. This seemingly weaker condition in fact suffices to provide a full annotation of the loop (see Lemma 1).

Further Information This summary of pGCL is incomplete, and covers only those elements essential to the following exposition. For full details of the language and its formal semantics, the interested reader is directed to McIver and Morgan [2004], or for details of the mechanisation in Isabelle/HOL to Cock [2012, 2013].

2 The Proof

To model the system, we fix two distributions: the prior distribution on secrets, $P[S]$, and the conditional distribution of observation-given-secret, $P[o'|S]$. We assume throughout that these are valid distributions:

$$\sum S{\in}UNIV.\ P[S] = 1 \qquad 0 \leq P[s]$$
$$\sum o'{\in}UNIV.\ P[o'|S] = 1 \qquad 0 \leq P[S|o']$$

We capture these, and the assumption that both the space of secrets and of observations is finite, in an Isabelle local theory (locale).

2.1 Modelling the Guessing Attack.

The state of the attack after some number of guesses (perhaps 0) is simply the list of observations that the attacker has made so far, or $\mathcal{O}$. In order to capture the system's (probabilistic) choice of secret, and the attacker's choice of strategy, however, we add two extra state components: $\mathcal{S}$ — the secret, and τ — the strategy. The system state is thus represented by the following record[3]:

record $('s, 'o)$ *attack-state* =
  $\mathcal{S} :: 's$
  $\mathcal{O} :: 'o\ list$
  $\tau :: 'o\ list \Rightarrow 's$

The use of a record as the state type allows us to take advantage of the field-update syntax introduced in our previous work[Cock, 2013].

We model the attack as a loop, with the attacker making one guess and one observation per iteration. This specification of a single iteration of the loop body demonstrates the syntax for probabilistically binding a name (without updating the state), and updating a single state component as if it were a variable:

[3] A tuple with named fields

$$
\begin{array}{l}
\mathit{body} \equiv \\
\quad \mathit{bind\ obs\ at\ } (\lambda s.\ P[\mathit{obs}|\mathcal{S}\ s])\ \mathit{in} \\
\quad \mathcal{O} := (\lambda s.\ \mathit{obs} \cdot (\mathcal{O}\ s))
\end{array}
$$

The full attack has three phases: first the attacker decides on a strategy, knowing both the distribution on secrets ($P[S]$), and the conditional probability of observations, given secrets ($P[o'|S]$). This is expressed as an unconstrained nondeterministic choice (*any*), in a context where the names *P-s* and *P-os* are already bound (these are the underlying constants corresponding to the syntax $P[S]$ and $P[o'|S]$).

Next, the system generates a secret at random, according to the distribution $P[S]$, expressed using a *probabilistically-constrained* choice[4]. It is critical that this step comes *after* the first, as otherwise the attacker could simply choose a strategy where $\tau\ s\ [] = \mathcal{S}\ s$. Semantically, we avoid this as the value $\mathcal{S}\ s$ is only bound *after* the choice of τ.

Finally, starting with an empty observation list, the attacker probes the system, collecting observations, until it terminates by guessing correctly. Note that there is no guarantee that this loop terminates, for example if the attacker repeats an incorrect guess again and again. We will detail shortly how this affects the formalisation, as it interacts with our definition of vulnerability.

The full attack is expressed in pGCL as follows:

$$
\begin{array}{l}
\mathit{attack} = \\
\quad \mathit{any}\ \tau\ ;; \\
\quad \mathit{any}\ \mathcal{S}\ \mathit{at}\ (\lambda s\ S.\ P[S])\ ;; \\
\quad \mathcal{O} := []\ ;; \\
\quad \mathit{do}\ G \longrightarrow \mathit{body}\ \mathit{od}
\end{array}
$$

where

$$G\ s = (\tau\ s\ (\mathcal{O}\ s) \neq \mathcal{S}\ s)$$

We now need a security predicate: a judgement on states, declaring them either secure or insecure. While a predicate in pGCL can take any (non-negative) real value, we begin by cleanly distinguishing secure states. Our security predicate is thus the embedding of a boolean predicate, and therefore takes only the values 0 (for insecure) and 1 (for secure). Under the action of the above program (interpreted as an *expectation transformer*), this 0,1-valued predicate (in the postcondition) is transformed to its *expected value*: the value of the weakest pre-expectation is (a lower bound on) the *probability* that we end in a secure state. The complement is therefore the *probability of compromise*, if the system begins in that state.

[4] The alternatives in a probabilistic choice need not sum to 1: the remainder of the probability is assigned to a nondeterministic choice among all possibilities. Thus, any branch is taken with *at least* the specified probability, and perhaps more. In this way, nondeterministic choice is a special case of (sub-)probabilistic choice, where all branches have probability 0.

We consider the system to be secure in its final state if the attacker has guessed correctly: $\tau\ s\ (\mathcal{O}\ s) = \mathcal{S}\ s$, but has taken at least n incorrect guesses to do so: $n < |\mathcal{O}\ s| \wedge (\forall i{\leq}n.\ \tau\ s\ (tail\ i\ (\mathcal{O}\ s)) \neq \mathcal{S}\ s)$.

The first of these conditions may appear odd, in that we are asserting that the system is only secure if the attacker knows the secret! The technical reason is that this term is the negation of the loop guard, G. The trick, and the reason that we can safely include this conjunct, is that this predicate is only applied to *final* states i.e those where the loop has terminated, which, by definition, only occurs once the guard is invalidated. This term thus only ever appears in a conjuction with the guard, leaving us with a term of the form $\llparenthesis\ G\ \rrparenthesis\ s * \llparenthesis\ G\ \rrparenthesis\ s * x$, which simply collapses to $\llparenthesis\ G\ \rrparenthesis\ s * x$, the security predicate.

If the attacker has not yet guessed correctly, then our definition of security depends not only on this predicate, but also on the interpretation on non-terminating programs, which we will address shortly.

$$
\begin{array}{l}
secure\ n\ s = \\
\llparenthesis\ \lambda s.\ \tau\ s\ (\mathcal{O}\ s) = \mathcal{S}\ s\ \rrparenthesis\ s\ * \\
\llparenthesis\ \lambda s.\ n < |\mathcal{O}\ s| \wedge (\forall i{\leq}n.\ \tau\ s\ (tail\ i\ (\mathcal{O}\ s)) \neq \mathcal{S}\ s)\ \rrparenthesis\ s
\end{array}
$$

Finally, we define vulnerability as the complement of security: the prior vulnerability is the probability that the system will end in an insecure state, given at most (n) guesses. This is, of course, the complement of the probability that the system ends in a *secure* state, thus the definition:

$$V_n\ n = 1 - wlp\ attack\ (secure\ n)\ (SOME\ x.\ True)$$

The term $SOME\ x.\ True$ is the Hilbert choice on the universal set. This simply expresses our intention that the vulnerability does not depend on the initial state.

Our definition of vulnerability is made in this slightly roundabout fashion ('the complement of the likelihood of success', rather than simply 'the likelihood of failure'), in order to ensure that it is preserved under *refinement*.

It is a well-recognised problem[Morgan, 2006] that many security properties (for example noninterference) are not preserved under refinement: a nondeterministic choice is refined by any of its branches, in particular, a nondeterministic choice of guess, is refined by the program that simply guesses the correct secret on the first try. The same problem does not occur with probabilistic choice in pGCL: it is already maximal in the refinement order. This is clearer on inspecting the destruction rule for refinement:

$$\frac{prog \sqsubseteq prog' \qquad nneg\ P}{wp\ prog\ P\ s \leq wp\ prog'\ P\ s}$$

Here we see that for any non-negative expectation P, and any initial state s, a refinement assigns at least as high a value (or probability, in the case of a boolean post-expectation) to the pre-expectation as the original program did. Thus, by phrasing our postcondition as 'the system is secure', any refinement will only

increase the probability of remaining secure, and thus decrease vulnerability. If we instead chose 'the system is compromised', then refinement would work against us: a refinement could have greater vulnerability than its specification.

The final wrinkle is that, while refinement now acts in the 'right' direction, nontermination doesn't. Under the strict semantics (wp), the weakest pre-expectation of *any* postcondition in an initial state from which the program diverges is 0—the probability of terminating *and* establishing the postcondition is zero. The solution to this dilemma is to switch to the liberal semantics (wlp). Here we ask for the probability of establishing the postcondition *if* terminating, rather then establishing it *and* terminating, as in the strict case. The only difference between the two is in the treatment of non-terminating programs. In particular the refinement order on terminating programs is unaffected, and thus our property is still preserved by refinement. This choice also matches our intuitive expectation: if the attacker never guesses the secret (either because it repeats failed guesses, or because either party fails or enters an infinite loop and stops responding), the system is secure.

Having established that the model meets our expectations, we turn to the second of the two instances in which cleverness is required: annotating the loop.

2.2 Annotating the Attack Loop

Annotating a probabilistic loop is very similar to annotating a classical loop: Any invariant, combined with the guard, gives an annotation, via the loop rule:

Lemma 1. *If the expectation I is an* invariant *of the loop $do\ G \longrightarrow body\ od$:*

$$« G » \, s * I\, s \leq wlp\ body\ I\ s$$

then the following annotation holds for the loop itself:

$$I\, s \leq wlp\ do\ G \longrightarrow body\ od\ (\lambda s.\ « \mathcal{N}\ G » \, s * I\, s)\ s \tag{2}$$

Proof. See McIver and Morgan [2004]

The verification challenge is also the same: to find an invariant that is simultaneously strong enough to imply the desired postcondition, and weak enough to be established by the given precondition. Here, we take a well-known tactic from conventional invariant selection, and modify it to suit a probabilistic setting.

The trick is to split the invariant into a conjunction of two parts: a predicate that represents the intended postcondition, as it would look in the current state, and a separate predicate that 'prunes' the set of future traces. The first half is chosen so that it evaluates to the postcondition (in our case the security predicate) in any terminating state, and the second to only allow traces that preserve the first half. Given that we are manipulating probabilities, and not truth values, we use a product rather than a conjunction. The result is equivalent,

as a glance at the truth table will demonstrate[5]. The first 'conjunct' (the portion on the left-hand side of the product) is thus logically equivalent to our security predicate.

The right conjunct is rather different, however. Rather than choosing a predicate that prunes the undesired traces, we instead take the weighted sum over all traces (lists of observations), of the *probability* that the given trace preserves the first conjunct. The weight assigned to a trace is in turn the probability of it occurring which, by the assumption of independence (Equation 1), is simply the product of the conditional probability of each observation, given the secret. We thus construct our invariant as *the sum over all possible futures, weighted by the probability of establishing the postcondition.*

Note that the syntax $\sum l[..n].\ f\ l$ refers to the sum over all lists of length n, and $R\ b$ to the embedding of a boolean value as either 0 or 1:

$$
\begin{array}{l}
I\ n\ s = \\
\quad (\prod i{=}0..min\ n\ |\mathcal{O}\ s|.\ R\ (\tau\ s\ (tail\ i\ (\mathcal{O}\ s)) \neq \mathcal{S}\ s)) \\
* \ (\sum ol[..n - |\mathcal{O}\ s|\ ]. \\
\quad \prod i = |\mathcal{O}\ s| + 1\ ..\ n. \\
\quad\quad P[((ol\ @\ (\mathcal{O}\ s))\ !\ (n{-}i))|(\mathcal{S}\ s)] \\
\quad * \ R\ (\tau\ s\ (tail\ i\ (ol\ @\ (\mathcal{O}\ s))) \neq \mathcal{S}\ s))
\end{array}
$$

The proof that this is indeed an invariant is straightforward (with full details of this and all other proofs available in the accompanying theory source).

Lemma 2. *The expectation I is an invariant of the loop $do\ G \longrightarrow body\ od$ i.e.:*

$$\llbracket\ G\ \rrbracket\ s * I\ n\ s \leq wlp\ body\ (I\ n)\ s$$

Proof. By unfolding. □

Moreover, the combination of the invariant and the guard is precisely the security predicate:

$$
\begin{array}{l}
\llbracket\ \mathcal{N}\ G\ \rrbracket\ s * I\ n\ s = \\
\llbracket\ \lambda s.\ \tau\ s\ (\mathcal{O}\ s) = \mathcal{S}\ s\ \rrbracket\ s\ * \\
\llbracket\ \lambda s.\ n < |\mathcal{O}\ s| \wedge (\forall\, i{\leq}n.\ \tau\ s\ (tail\ i\ (\mathcal{O}\ s)) \neq \mathcal{S}\ s)\ \rrbracket\ s
\end{array}
$$

Thus, applying Lemma 1, we have:

$$I\ n\ s \leq wlp\ do\ G \longrightarrow body\ od\ (secure\ n)\ s \tag{3}$$

[5] Note that multiplication is not the only choice that preserves boolean conjunction: the 'probabilistic conjunction' $p\ .\&\ q \equiv max\ 0\ (p + q - 1)$ would also work, and is used in other parts of the pGCL formalisation. The reason for instead choosing multiplication is that we will later manipulate it algebraically to create a product of probabilities.

2.3 Annotating the Initialisation

The loop initialisation (assigning the empty list of observations) is annotated just as in a classical setting, by instantiating the pre-expectation, and thus:

Lemma 3 (Initialisation).

$$\begin{array}{l}\sum ol[..n].\ (\prod i = 1..n.\ P[ol_{[n\ -\ i]}|\mathcal{S}\ s]) * \\ \qquad (\prod i = 0..n.\ R\ (\tau\ s\ (tail\ i\ ol) \neq \mathcal{S}\ s)) \\ \leq wlp\ (\mathcal{O} := [];; \\ \qquad do\ G \longrightarrow body\ od) \\ \quad (secure\ n)\ s\end{array}$$

Proof. By instantiating Equation 3, we have:

$$\begin{array}{l}R\ (\tau\ s\ [] \neq \mathcal{S}\ s) * \\ (\sum ol[..n].\ \prod i = 1..n.\ P[ol_{[n\ -\ i]}|\mathcal{S}\ s] * R\ (\tau\ s\ (tail\ i\ ol) \neq \mathcal{S}\ s)) \\ \leq wlp\ (\mathcal{O} := [];; \\ \qquad do\ G \longrightarrow body\ od) \\ \quad (secure\ n)\ s\end{array}$$

whence we rearrange the pre-expectation by splitting the product and distributing over the summation:

$$\begin{array}{l}R\ (\tau\ s\ [] \neq \mathcal{S}\ s) * \\ (\sum ol[..n].\ \prod i = 1..n.\ P[ol_{[n\ -\ i]}|\mathcal{S}\ s] * R\ (\tau\ s\ (tail\ i\ ol) \neq \mathcal{S}\ s)) = \\ (\sum ol[..n].\ (\prod i = 1..n.\ P[ol_{[n\ -\ i]}|\mathcal{S}\ s]) * \\ \qquad (\prod i = 0..n.\ R\ (\tau\ s\ (tail\ i\ ol) \neq \mathcal{S}\ s)))\end{array}$$

at which point the result follows. □

Making the choice over secrets, this becomes the weighted sum:

$$\begin{array}{l}\sum S.\ P[S] * \\ \quad (\sum ol[..n].\ (\prod i = 1..n.\ P[ol_{[n\ -\ i]}|S]) * \\ \qquad (\prod i = 0..n.\ R\ (\tau\ s\ (tail\ i\ ol) \neq S)))\end{array}$$

We manipulate this expression to produce a sum of probabilities, by first defining the *guess set* (Γ) of a strategy—the set of all guesses produced by terminal sublists of the given list of observations:

$$\begin{array}{l}\sigma_{tr}\ \sigma\ [] = [([], \sigma\ [])]\ | \\ \sigma_{tr}\ \sigma\ (o' \cdot os) = (o' \cdot os, \sigma\ (o' \cdot os)) \cdot \sigma_{tr}\ \sigma\ os \\ \Gamma\ \sigma\ ol = snd\ `\ set\ (\sigma_{tr}\ \sigma\ ol)\end{array}$$

We can now annotate the entire attack, including the nondeterministic choice of strategy:

Lemma 4 (The Attack). *We have:*

$(\bigsqcap x.\ 1 - (\sum ol[..n].\ \sum S{\in}\Gamma\ x\ ol.\ P[ol,S])) \leq wlp\ attack\ (secure\ n)\ s$

Proof. *We begin by noting that the pre-expectation of the choice over secrets:*

$$\sum S.\ P[S] * (\sum ol[..n].\ (\prod i = 1..n.\ P[ol_{[n\ -\ i]}|S]) * (\prod i = 0..n.\ R\ (\sigma\ (tail\ i\ ol) \neq S)))$$

can be rewritten (by changing the order of summation, and distributing multiplication) to:

$$\sum ol[..n].\ \sum S.\ P[S] * (\prod i = 1..n.\ P[ol_{[n\ -\ i]}|S]) * (\prod i = 0..n.\ R\ (\sigma\ (tail\ i\ ol) \neq S))$$

We then note that the innermost product is simply the joint probability of the secret and the list of observations, and thus we have:

$\sum ol[..n].\ \sum S.\ P[ol,S] * (\prod i = 0..n.\ R\ (\sigma\ (tail\ i\ ol) \neq S))$

Finally, we note that by the definition of Γ*, if* $|ol| = n$ *then*

$((\prod i = 0..n.\ R\ (\sigma\ (tail\ i\ ol) \neq S)) = 0) = (S \in \Gamma\ \sigma\ ol)$

and thus we have

$1 - (\sum ol[..n].\ \sum S{\in}\Gamma\ \sigma\ ol.\ P[ol,S])$

as the reworked precondition for the choice over secrets. The result then follows from the definition of nondeterministic choice as the infimum *over the branches.* □

2.4 The Top-Level Theorem

Our ultimate result then follows:

Theorem 1. *The vulnerability of the system is bounded above by a sum over the joint probability of the full list of observations, and the set of the attacker's guesses for all initial sublists of observations (intermediate states), maximised over possible choices of strategy.*

$V_n\ n \leq (\bigsqcup x.\ \sum ol[..n].\ \sum S{\in}\Gamma\ x\ ol.\ P[ol,S])$

Proof. *Follows from Lemma 4, the definition of* V_n *and the algebraic properties of the infimum.* □

From this theorem, a number of facts are immediately apparent. First, as every term in the sum is non-negative, vulnerability is (weakly) monotonic in the size of the guess set. Thus (assuming that there are enough distinct secrets for it to be possible), a strategy that repeats can be extended into a *non-repeating* strategy (by replacing repeated guesses with fresh secrets), with at least as high a

vulnerability. Thus we need only consider non-repeating strategies in calculating the supremum.

Secondly, and more importantly, for any strategy, vulnerability is a sum of the joint probability of a list of observations and a guess. It is thus obvious that a strategy that maximises this, also maximises vulnerability. This is, of course, the MAP strategy that we introduced in Section 1.2. In fact, the above prohibition on repeating strategies can be rephrased in terms of distributions: the optimal strategy maximises the joint probability over the distribution where the probability of any secret that has already been guessed is set to zero (and the rest scaled appropriately).

We have now come full circle, eliminating our operational model entirely, and finishing with a formulation solely in terms of the statistical properties of the system that, as we see, matches our expectations. Importantly, we didn't *assume* anything about the optimality of the non-repeating MAP strategy: it fell out as a *consequence* of the model.

3 Using the Result

We conclude with a sketch to illustrate how this result fits into the overall theory, and provides the connection between the operational and information-theoretic models.

As indicated at the end of Section 2, the pen-and-paper proof picks up at Theorem 1, establishing the optimality of the MAP strategy. Given this, we are able to establish a number of results extant in the literature. For example V_1, or the chance of compromise in a single guess, is simply the probability of the most likely secret:

$$\begin{aligned} V_1 &= \max_S P[[\,], S] \\ &= \max_S P[S] \end{aligned}$$

This is closely related to the *min-entropy* measure (H_∞), which has recently begun to supplant Shannon entropy in the formal analysis of vulnerability to information leakage[Smith, 2009]. Specifically:

$$H_\infty(P) = -\log_2(\max_S P[S]) = -\log_2 V_1$$

From this, we synthesise a *leakage measure*, or a bound on how vulnerability changes over time, by analogy with the min-leakage: $L_\infty = H_\infty(P_1) - H_\infty(P_2)$. We estimate the rate of leakage by the ratio of the vulnerability given 2 guesses, to that given only 1. This multiplicative measure becomes additive, once we

rephrase it in terms of entropy:

$$\begin{aligned} L &= \frac{V_2}{V_1} \\ \log_2 L &= \log_2 \frac{V_2}{V_1} \\ &= \log_2 V_2 - \log_2 V_1 \\ &= H_\infty(P_1) - H_\infty(P_2) \end{aligned}$$

Ultimately, we extend the model to allow nondeterministic choice not only over strategies, but also over the distributions $P[S]$ and $P[o|S]$ themselves, taken from two sets, Q_S and Q_{oS} respectively. Thanks to our definition of vulnerability, this is equivalent to asking for the greatest (strictly, supremum) vulnerability over all distributions in Q_S and Q_{oS}.

A particularly interesting case occurs when Q_S is the set of all distributions of *Shannon* entropy H_1, and Q_{oS} the set of all conditional distributions with *channel capacity* C. We thus express the worst case vulnerability, given that we only know the Shannon entropy of the distribution (this is not generally straightforward to calculate, see Smith [2009]). Moreover, on average (assuming that the space of secrets is large enough that the effect of the individual yes/no answers is small), the entropy remaining after a single guess is just $H - C$. We thus iterate the model, recalculating V_n, this time setting Q_S to the set of all distributions of entropy $H - C$.

Finally, consider what happens to the vulnerability as we vary Q_S. As the size of the set increases, the supremum is taken over more possibilities, and thus we should expect the vulnerability to increase. Likewise, smaller sets should give lower vulnerability. This is indeed precisely what we see as, from the semantics of nondeterminism in pGCL, the choice over some $S \subseteq T$ is a refinement of the choice over T. Recall that a refinement assigns a *higher* value to every state than the original program and thus, as it appears negated in our definition of vulnerability, the choice over a smaller set gives a *lower* vulnerability, and vice versa.

We thus see that the order on the vulnerability bounds is determined by the subset order on the set of distributions. In particular, this gives us a complete lattice of bounds, since $V_1\ \emptyset = 0$ (by definition) and $V_1\ \top = 1$ (this includes every distribution that assigns 100% probability to a single secret, which the attacker will then guess with certainty). We thus link the refinement order on our operational models, with a (dual) order on this complete lattice of bounds. For the full derivation see Cock [2014].

4 Related Work

This work draws on our own previous work on the mechanisation of probabilistic reasoning, together with results in both the programming-language semantics and the security literature.

We build on our own previous work in mechanising[Cock, 2012] the pGCL semantics of McIver and Morgan [2004], and in demonstrating the feasibility of carrying probabilistic results down to real systems by incorporating large existing proof results[Cock, 2013, Klein et al., 2009]. We extend this by demonstrating that we can take the approach one step further: into the domain of information theory.

The concept of entropy, and the solution to the optimal decoding problem both date to the earliest years of information theory[Shannon, 1948], while the shift to *min*-entropy in the security domain largely follows the insight of Smith [2009] that the single-guess vulnerability of a distribution is only loosely connected to its Shannon entropy. The further generalisation of this idea is a subject of active research[Alvim et al., 2012, Espinoza and Smith, 2013, McIver et al., 2010]. Our own previous work include the rigorous evaluation of the *guessing attack* as a threat model[Cock, 2014].

Acknowledgements

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

Bibliography

Mário S. Alvim, Kostas Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In *25th CSF*, pages 265–279. IEEE, 2012. doi: 10.1109/CSF.2012.26.

David Cock. Verifying probabilistic correctness in Isabelle with pGCL. In *7th SSV*, pages 1–10, Sydney, Australia, Nov 2012. doi: 10.4204/EPTCS.102.15.

David Cock. Practical probability: Applying pGCL to lattice scheduling. In *4th ITP*, pages 1–16, Rennes, France, Jul 2013. doi: 10.1007/978-3-642-39634-2_23.

David Cock. *Rigorous Approaches to Side Channels in Componentised Systems*. PhD thesis, School Comp. Sci. & Engin., Sydney, Australia, 2014.

Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8):453–457, Aug 1975. ISSN 0001-0782. doi: 10.1145/360933.360975.

Barbara Espinoza and Geoffrey Smith. Min-entropy as a resource. *Inform. & Comput.*, 226:57–75, 2013. ISSN 0890-5401. doi: 10.1016/j.ic.2013.03.005.

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *SOSP*, pages 207–220, Big Sky, MT, USA, Oct 2009. ACM. doi: 10.1145/1629575.1629596.

Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2004. ISBN 978-0-387-40115-7. doi: 10.1007/b138392.

Annabelle McIver, Larissa Meinicke, and Carroll Morgan. Compositional closure for bayes risk in probabilistic noninterference. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and PaulG. Spirakis, editors, *Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 223–235. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-14161-4. doi: 10.1007/978-3-642-14162-1_19.

Carroll Morgan. The shadow knows: Refinement of ignorance in sequential programs. In Tarmo Uustalu, editor, *Mathematics of Program Construction*, volume 4014 of *Lecture Notes in Computer Science*, pages 359–378. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-35631-8. doi: 10.1007/11783596_21.

Claude E. Shannon. A mathematical theory of communication. *The Bell Syst. Techn. J.*, 1948. doi: 10.1145/584091.584093. Reprinted in SIGMOBILE Mobile Computing and Communications Review, 5(1):3–55, 2001.

Geoffrey Smith. On the foundations of quantitative information flow. In *12th FOSSACS*, pages 288–302, York, UK, 2009. Springer. doi: 10.1007/978-3-642-00596-1_21.