

Formalized Information-Theoretic Proofs of Privacy using the HOL4 Theorem-Prover

Aaron R. Coble

University of Cambridge Computer Laboratory, 15 JJ Thomson Avenue, Cambridge
CB3 0FD, UK arc54@cam.ac.uk

Abstract. Below we present an information-theoretic method for proving the amount of information leaked by programs formalized using the HOL4 theorem-prover. The advantages of this approach are that the analysis is quantitative, and therefore capable of expressing partial leakage, and that proofs are performed using the HOL4 theorem-prover, and are therefore guaranteed to be logically and mathematically consistent with the formalization. The applicability of this methodology to proving privacy properties of Privacy Enhancing Technologies is demonstrated by proving the anonymity of the Dining Cryptographers protocol. To the best of the author’s knowledge, this is the first machine-verified proof of privacy of the Dining Cryptographers protocol for an unbounded number of participants and a quantitative metric for privacy.

1 Introduction

Gene Spafford once said, “The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards – and even then I have my doubts.” [11] While some, if not most, of the security community will agree with this sentiment, it does not negate the desire to develop technologies that increase the security of systems and methods to analyze and quantify the relative security of these technologies. As Privacy-Enhancing Technologies (PETs) are developed, the need arises for methods to analyze and quantify the relative privacy guarantees of various PETs. It is important that these methods of analysis are quantitative (rather than a boolean result of “secure” or “insecure”) because many deployable PETs must make concessions for the sake of efficiency and do not guarantee absolute privacy. Pioneering work done independently by both Serjantov and Danezis [26] and Díaz [12] proposed the use of entropy as a metric for privacy, thereby linking Shannon’s Information Theory [27] to quantitative analysis of privacy and giving rise to a subsequent branch of privacy analysis. Since then most quantitative privacy analysis is rooted in Information Theory. More recent work in this area e.g. Chatzikokolakis’ use of Bayes Risk [3], has offered a variety of related metrics for privacy based on Information Theory.

Historically, various products of formal methods research, such as theorem-provers and model-checkers, have been fruitfully applied to security analysis.

Paulson developed the technique of using a theorem-prover to formalize security protocols and perform inductive proofs of security for these protocols [22]. Model-checking has been widely used both for finding bugs in security protocols and for verifying finite instances of security protocols; Lowe’s use of FDR to find a bug in the Needham-Schroeder protocol [20] is a well known example. Each of these analysis techniques has its relative advantages and disadvantages. Model-checking requires less human effort because it is fully automatic, once a system and its desired properties are formalized. However, model-checking is limited to (generally) small, finite instances of a system due to a state-explosion which renders larger instances intractable. Clearly verification of a particular instance of the system does not necessarily provide any guarantees about the system generally. This makes model-checking techniques more applicable for bug-finding in security systems than for proving security guarantees. Theorem-proving usually requires a greater amount of human effort than model-checking because proofs are not automatic and must be performed through interaction with the theorem-prover. The advantage of theorem-proving is that proofs can be quantified on the parameters of the system, allowing for a proof of security for the system generally rather than for a particular finite instance. This advantage of theorem proving is exemplified by the proof of privacy of the dining cryptographers protocol discussed in Section 3, which is valid for an *unbounded number* of protocol participants. While theorem-proving is aimed at proof of correctness rather than bug-finding, an unsuccessful proof attempt will provide insight into the reason that the system fails. The further advantage of interactive theorem-proving over pen-and-paper proofs is that proofs are guaranteed to be correct up to the assumptions of the model. (Recent work by Blanchet [2] has focused on bridging the gap between the computational-complexity proofs used by cryptographers and the assumption of perfect cryptographic-primitives commonly used in formal-methods proofs). We do not to claim that a machine-assisted proof is an absolute guarantee of security; Any proof of security is only valid up to the level of abstraction at which the system is modeled – a point widely noted in the literature where security and formal methods research intersect. However, LCF-style theorem provers, such as the HOL4 system used in this work, do guarantee the logical consistency of all proofs they produce[16]. This is achieved by using a small logical core from which all theorems must be derived using basic inferences rules. A substantial amount of mathematical theory, including probability theory, which this work makes use of, has been previously been formalized in HOL4. Since pen-and-paper proofs are often long, complicated, and prone to errors, the guarantee of correctness provided by using a theorem-prover is valuable, particularly for security applications.

Model-checking techniques have been successfully used to analyze various PETs e.g. Shmatikov et al.’s use of PRISM [28] to analyze the Crowds protocol[23]. However, the author is unaware of any prior work applying theorem-proving techniques to quantitative privacy analysis; this paper aims to begin filling that void. In spirit this work belongs both to the branch of PETs research begun by Serjantov, Danezis, and Díaz and to the branch of formal methods work begun

by Paulson. Recent work by Malacaria et al. proposed an Information-Theoretic approach to analyzing the information leakage in programs [21] and this work serves as inspiration for the formalized analysis technique presented in this paper. We will now move on to examine a formalization of Shannon’s Information Theory in Higher Order Logic (HOL) using the HOL4 theorem-prover and the use of this formalization for quantifying the information leakage of programs. Afterwards, the applicability of this technique to analysis of PETs is demonstrated using the Dining Cryptographers problem [4] as a case study. To the best of the author’s knowledge, this is the first machine-verified proof of privacy of the Dining Cryptographers protocol for an *unbounded number* of participants and a *quantitative* metric for privacy. Finally, future directions and applications for this work will be presented.

2 Formal Analysis of Information Leakage using HOL4

We will now examine a formalization of Shannon’s Information Theory in HOL4. Basic concepts from information theory and probability theory will be explained concurrently with their formalization for the benefit of the unfamiliar reader. Hurd’s formalization of probability theory in HOL4 [18] serves as a starting point for our formalization. In [18], Hurd formalized measure theory in HOL4 and building on this he formalized a definition for probability spaces and functions on these. (See [14] for an excellent introduction to how probability theory is derived from measure theory). Hurd then used this formalization to verify the correctness of probabilistic algorithms. While the work presented in [18] was a major milestone towards machine-verification of probabilistic algorithms, the scope of that work had its limits. Important results for *independent* functions on probability spaces were developed and theorems about the properties of a probability measure were proved (e.g. the probability of the empty event is 0, the probability of the universe of all measurable events is 1, and a probability measure is countably additive); however, many concepts from probability theory such as definitions for random variables, the expected value of random variables, and conditional probability do not feature in [18]. In [17], Hasan built upon Hurd’s work to define the expected value of and prove properties of some common discrete probability distributions including the bernoulli distribution. Unfortunately Hasan’s work is not directly applicable for the work presented here, so we must begin where Hurd’s formalization ends by defining random variables and their expectation.

2.1 Information Theory formalized in HOL4

We will now derive a formalization of Information Theory in HOL from basic elements of probability theory. First we recall textbook definitions for a *probability space* and measurable *events* in this space. (The unfamiliar reader is referred to [14, 29] or similar introductory texts for further details). The formalization of these two notions is part of the work in [18].

Definition 1. Let $\mathcal{P}(\text{outcomes})$ be the powerset of all possible outcomes of some experiment. For a set $\Omega \subseteq \mathcal{P}(\text{outcomes})$ and function μ from elements of Ω to \mathbb{R} , (Ω, μ) defines a **probability space** iff (i) Ω is closed under countable-unions, -intersections, and -complementations, (ii) Ω contains the universe of all outcomes UNIV and the empty set $\{\}$, (iii) $\forall x \in \Omega. 0 \leq \mu(x) \leq 1$, (iv) $\mu(\text{UNIV}) = 1$ and $\mu(\{\}) = 0$, and (v) μ is countably-additive on Ω .

Definition 2. For a probability space (Ω, μ) , each element of Ω is an **event**. Each event denotes a set of outcomes for an experiment. For example, let $\mathcal{E} = \{\text{saw 1 red bird, saw 1 blue bird}\}$ defines an event \approx “saw 1 bird”. Events can be combined using the usual set operations (Union, Intersection, Complementation, Difference) to define other events.

We now move on to review a textbook definition of a *random variable* on a probability space and to develop our formalization of this definition in HOL.

Definition 3. For a measurable space with measurable sets Ψ of type $: ((\beta \text{ set}) \text{ set})$ and a probability space (Ω, μ) , with sample space Ω of type $: ((\alpha \text{ set}) \text{ set})$ and probability measure μ of type $: (\alpha \text{ set} \rightarrow \mathbb{R})$, a **random variable** \mathcal{X} is a map of type $: (\alpha \rightarrow \beta)$ s.t. $\forall y. y \in \Psi \Rightarrow \{x \mid \mathcal{X}(x) \in y\} \in \Omega$ i.e. a random variable is measurable function from the sample space of a probability space to the measurable sets of a measurable space. The probability of \mathcal{X} taking the value x is defined as $P(\mathcal{X} = x) = \mu \{y \mid \mathcal{X}(y) \in x\}$.

Typically, random variables are real- or natural-valued functions; however we will use the more general definition for our formalization. We can formalize the definition of a random variable and its probability measure in HOL as

$$\text{random_variable } (s, \mu) \ X = \\ \text{prob_space } (s, \mu) \ \wedge \ \text{measurable } X \ s \ \text{UNIV}$$

and

$$\text{random_variable_prob } (s, \mu) \ X = \lambda x. \mu \{y \mid X(y) \text{ IN } x\}.$$

The attentive reader will have noticed that we are requiring a random variable to be measurable on the universe of a type rather than some subset thereof. This is because Ψ from Definition 3 must contain all of the singleton sets in order for the *expected value* of a random variable to be well defined; any Ψ containing all of the singleton sets is necessarily the universe because it must be closed under countable unions in order to be measurable. This condition is clearly satisfied in the typical case where Ψ is defined by the space of the real or natural numbers.

For the present discussion, we are restricting the probability distribution μ to be a *discrete* probability distribution. This does not impose any real restriction on our formalization as we are interested in statements of the form, “What is the probability that the sender of this message was Bob?”, $P(\text{sender} = \text{Bob})$, rather than statements of the form “What is the probability that the Bob’s weight is less than 100kg?”, $P(\text{weight} < 100)$. In the interest of brevity, we will also limit

the present discussion to *finite* random variables i.e. those X and (s, μ) for which

$$\{x \mid \text{random_variable_prob } (s, \mu) X \{x\} \neq 0\}$$

is *finite*; formalization of *countable* random variables i.e. those for which the above set is *countable* will not be presented here, since finite random variables suffice for the application of interest.

We are often interested in a *vector* of random variables, $\mathcal{X}_1, \dots, \mathcal{X}_n$, rather than a single random variable; the joint probability measure of such a vector is the natural one $P(\mathcal{X}_1 = x_1, \dots, \mathcal{X}_n = x_n) = \mu \{y \mid \mathcal{X}_1(y) \in x_1 \wedge \dots \wedge \mathcal{X}_n(y) \in x_n\}$. A formalization of random vectors and their probability measures can be defined using a list to represent the vector of random variables as follows

$$\begin{aligned} \text{random_vector } (s, \mu) [X] &= \text{random_variable } (s, \mu) X \\ \text{random_vector } (s, \mu) (X :: XS) &= \text{random_variable } (s, \mu) X \wedge \\ &\quad \text{random_vector } (s, \mu) XS \end{aligned}$$

and

$$\begin{aligned} \text{random_vector_prob } (s, \mu) XS &= \\ \lambda \text{ xs. } \mu \{y \mid \text{FOLDR } \wedge \top (\text{MAP } (\lambda b. (\text{FST } b) \text{ IN } (\text{SND } b)) \\ &\quad (\text{ZIP } (\text{MAP } (\lambda X. X(y)) XS) \text{ xs}))\}, \end{aligned}$$

where $::$ is the list construction operation, MAP maps a function over a list, ZIP creates a list of pairs from two lists, FST and SND select the first and second element of a pair respectively, and FOLDR is the fold-right operation on a list. Note: we assume that xs is the same length as XS in order for $\text{random_vector_prob}$ to be sensibly defined above. For the sake of brevity, we will not present separate definitions for functions on random variables and random vectors where the extension is obvious.

Having formalized random variables and their probability measures, we now move on to review the definition of the *expected value* of a random variable.

Definition 4. *The expected value of a real-valued function f and a random variable (or random vector) \mathcal{X} is the average of the values taken by $(f \circ \mathcal{X})$ weighted by the probability measure of \mathcal{X} . The expected value is the most likely value for $(f \circ \mathcal{X})$ to take and is defined as*

$$\mathbb{E}(\mathcal{X}) = \sum_x (P(\mathcal{X} = x))(f(x)).$$

In order to formalize a definition for the expected value of finite random variables in HOL, we must first define a function SUM which is the summation of a real-valued function applied to the elements of a finite set. (Expected value for countable random variables is defined using a countably-infinite summation, but this formalization is outside the scope of this discussion). Due to its recursive definition, $\text{SUM } (f : \alpha \rightarrow \text{real}) (S : \alpha \text{ set})$ is well defined only when S is finite.

We then define a second HOL function $\text{sum } f \text{ S} = \text{SUM } f \{s \mid f(s) \neq 0\}$, which is the summation of f over those elements of S for which f takes a non-zero value. We can now formalize expected value in HOL as

$$\text{expected_value } (s, \mu) \text{ X } f = \\ \text{sum } \left(\lambda x. \left(\text{random_variable_prob } (s, \mu) \text{ X } \{x\} \right) (f(x)) \right) \text{ UNIV}$$

for random variables, or

$$\text{expected_value } (s, \mu) \text{ XS } f = \\ \text{sum } \left(\lambda xs. \left(\text{random_vector_prob } (s, \mu) \text{ XS } \left(\text{MAP } (\lambda x. \{x\}) \text{ xs} \right) (f(x)) \right) \right. \\ \left. \{xs \mid \text{LENGTH } xs = \text{LENGTH } XS\} \right)$$

for random vectors. (We allow this slight overloading as it is unlikely to cause confusion.)

We are now ready to develop a formalization of information theory in HOL from the formalizations for probability theory developed above. We begin by reviewing the definition of the *entropy* of a random variable and then proceed to formalize this definition in HOL. (Definitions for entropy and the other basic definitions from information theory reviewed below can be found in [27]).

Definition 5. *The entropy of a random variable (or random vector) \mathcal{X} captures the uniformity of the probability measure of \mathcal{X} i.e. the degree of uncertainty as to which outcome will occur. If \mathcal{X} takes a particular value with probability 1 then the entropy is 0. If the values of \mathcal{X} with non-zero probability are uniformly distributed (occur with equal probability) then the entropy is maximal.*

$$\mathbb{H}(\mathcal{X}) = - \sum_x P(\mathcal{X} = x) \log(P(\mathcal{X} = x)).$$

Building upon the formalization of expected value above, it is straightforward to formalize the definition of entropy in HOL as

$$\text{entropy } (s, \mu) \text{ X} = \\ \text{expected_value } (s, \mu) \text{ X} \\ (\lambda x. \text{lg } (\text{random_variable_prob } (s, \mu) \text{ X } \{x\}))$$

for random variables, or

$$\text{entropy } (s, \mu) \text{ XS} = \\ \text{expected_value } (s, \mu) \text{ XS} \\ (\lambda xs. \text{lg } (\text{random_vector_prob } (s, \mu) \text{ XS } (\text{MAP } (\lambda x. \{x\}) \text{ xs})))$$

for random vectors, using lg to abbreviate \log_2 . We choose to use a base of 2 for the logarithm in our formalization of entropy because of its correlation to *bits* of information; this is the typical choice for information-theoretic analysis. The reader is directed to [27] for a detailed explanation of this choice. We now move on to the information-theoretic concept of *conditional entropy*, first reviewing its definition and then developing its formalization in HOL.

Definition 6. The **conditional entropy** of random variable (or random vector) \mathcal{X} conditioned on the value of random variable \mathcal{Y} measures the uncertainty of \mathcal{X} given knowledge of \mathcal{Y} and is defined as

$$\mathbb{H}(\mathcal{X}|\mathcal{Y}) = \sum_y \mathbb{P}(\mathcal{Y} = y) \left(- \sum_x \mathbb{P}(\mathcal{X} = x | \mathcal{Y} = y) \log(\mathbb{P}(\mathcal{X} = x | \mathcal{Y} = y)) \right).$$

In order to formalize a definition for conditional entropy in HOL, we must first develop a formalization for the conditional probability measure of two random variables (or random vectors). We recall for the reader that the *conditional probability* of two random variables (or random vectors), \mathcal{X} and \mathcal{Y} , is $\mathbb{P}(\mathcal{X} = x | \mathcal{Y} = y) = \mathbb{P}(\mathcal{X} = x, \mathcal{Y} = y) / \mathbb{P}(\mathcal{Y} = y)$, where $\mathbb{P}(\mathcal{X} = x, \mathcal{Y} = y)$ is the joint probability measure of \mathcal{X} and \mathcal{Y} . The formalization of this definition in HOL is straightforward and as follows:

$$\begin{aligned} \text{random_variable_cond_prob } (s, \mu) \text{ X Y} = \\ \lambda (x, y). (\text{random_vector_prob } (s, \mu) \text{ [X; Y] [x; y]}) / \\ (\text{random_variable_prob } (s, \mu) \text{ Y } y) \end{aligned}$$

and

$$\begin{aligned} \text{random_vector_cond_prob } (s, \mu) \text{ XS YS} = \\ \lambda (xs, ys). (\text{random_vector_prob } (s, \mu) \text{ (XS++YS) (xs++ys)}) / \\ (\text{random_vector_prob } (s, \mu) \text{ YS } ys), \end{aligned}$$

where ++ is the list-concatenation operation. We now formalize conditional entropy in two steps, first defining something we will call conditioned entropy as

$$\begin{aligned} \text{conditioned_entropy } (s, \mu) \text{ X Y} = \\ \lambda y. - \text{sum } \left(\lambda x. (\text{random_variable_cond_prob } (s, \mu) \text{ X Y } (x, y)) \right. \\ \left. \lg(\text{random_variable_cond_prob } (s, \mu) \text{ X Y } (x, y)) \right) \text{ UNIV} \end{aligned}$$

for random variables, or

$$\begin{aligned} \text{conditioned_entropy } (s, \mu) \text{ XS YS} = \\ \lambda ys. - \text{sum } \left(\lambda xs. (\text{random_vector_cond_prob } (s, \mu) \text{ XS YS } (xs, ys)) \right. \\ \left. \lg(\text{random_vector_cond_prob } (s, \mu) \text{ XS YS } (xs, ys)) \right) \\ \{xs \mid \text{LENGTH } xs = \text{LENGTH XS}\} \end{aligned}$$

for random vectors, and finally defining conditional entropy as

$$\begin{aligned} \text{conditional_entropy } (s, \mu) \text{ X Y} = \\ \text{expected_value } (s, \mu) \text{ Y } (\text{conditioned_entropy } (s, \mu) \text{ X Y}). \end{aligned}$$

Building upon the formalizations developed above, we conclude this section by formalizing the definitions of *mutual information* and *conditional mutual information*; we first review the standard definition of each of these concepts before presenting its formalization in HOL.

Definition 7. *The correlation between two random variables (or random vectors) \mathcal{X} and \mathcal{Y} is captured by their **mutual information**, $\mathbb{I}(\mathcal{X}; \mathcal{Y})$. The greater their mutual information the greater the correlation between \mathcal{X} and \mathcal{Y} . Thus when \mathcal{X} and \mathcal{Y} are independent $\mathbb{I}(\mathcal{X}; \mathcal{Y}) = 0$. Mutual information measures the amount of information about \mathcal{X} that can be obtained by observing \mathcal{Y} (and vice versa).*

$$\mathbb{I}(\mathcal{X}; \mathcal{Y}) = \mathbb{H}(\mathcal{X}) - \mathbb{H}(\mathcal{X}|\mathcal{Y}) = \mathbb{H}(\mathcal{Y}) - \mathbb{H}(\mathcal{Y}|\mathcal{X}) = \mathbb{I}(\mathcal{Y}; \mathcal{X})$$

We now formalize mutual information in HOL as

$$\begin{aligned} \text{mutual_information } (s, \mu) \text{ X Y} = \\ \text{entropy } (s, \mu) \text{ Y} - \text{conditional_entropy } (s, \mu) \text{ X Y.} \end{aligned}$$

Definition 8. *The **conditional mutual information**, $\mathbb{I}(\mathcal{X}; \mathcal{Y}|\mathcal{Z})$, of random variables (or random vectors) \mathcal{X} and \mathcal{Y} conditioned on random variable (or random vector) \mathcal{Z} captures the correlation between \mathcal{X} and \mathcal{Y} given \mathcal{Z} . Conditional mutual information measures the amount of information about \mathcal{X} that can be learned by observing \mathcal{Y} (and vice versa) given knowledge of \mathcal{Z} .*

$$\mathbb{I}(\mathcal{X}; \mathcal{Y}|\mathcal{Z}) = \mathbb{H}(\mathcal{X}|\mathcal{Z}) - \mathbb{H}(\mathcal{X}|\mathcal{Y}, \mathcal{Z}) = \mathbb{H}(\mathcal{Y}|\mathcal{Z}) - \mathbb{H}(\mathcal{Y}|\mathcal{X}, \mathcal{Z}) = \mathbb{I}(\mathcal{Y}; \mathcal{X}|\mathcal{Z})$$

Conditional mutual information is formalized in HOL as

$$\begin{aligned} \text{conditional_mutual_information } (s, \mu) \text{ X Y Z} = \\ \text{conditional_entropy } (s, \mu) \text{ X Z} - \\ \text{conditional_entropy } (s, \mu) \text{ X (Y++Z).} \end{aligned}$$

2.2 Information Leakage Analysis

Having formalized a sufficient portion of information theory in HOL, we now proceed to develop an information-theoretic approach to analyzing the information leakage of programs using HOL. Intuitively, a program *leaks* information when an observer who knows (or can control) the low-security inputs to the program can learn or infer something about the high-security inputs to the program by observing the program's outputs. We recall from the previous section that conditional mutual-information of \mathcal{X} and \mathcal{Y} given \mathcal{Z} , $\mathbb{I}(\mathcal{X}; \mathcal{Y}|\mathcal{Z})$, measures the amount of information that can be learned about \mathcal{X} by observing \mathcal{Y} , given knowledge of \mathcal{Z} . If \mathcal{L} is a random variable ranging over the low-security inputs to a particular program, \mathcal{H} is a random variable over the high-security inputs, and \mathcal{O} is a random variable over the program's outputs, then $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L})$ (or equivalently $\mathbb{I}(\mathcal{H}; \mathcal{O}|\mathcal{L})$) measures the knowledge about the high-security inputs to the program that can be learned by observing the outputs of the program, given

knowledge of the low-security inputs. Borrowing from the work of Malacaria [21], Denning [9, 10], Clark [5], et al., we define the information leakage of a deterministic program to be $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L})$. For privacy (or anonymity) analysis, $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L})$ measures how much is learned about private (or identifying) information by an attacker who observes the outputs of a particular program and knows (or controls) any non-private inputs. In [6] Clark, et al. show that, for a deterministic program, the program is non-interfering iff $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L}) = 0$.

Programs to be analyzed are modeled in HOL as functions from low and high input states to output states. States are defined to be a polymorphic type, β *state*, s.t. states are functions $string \rightarrow \beta$, namely from variable names to values. Note that this allows us to have infinite state, e.g. $\beta = \mathbb{R}$; the only restriction we make, as noted earlier, is that probability distribution over states is non-zero for a finite number of states. Because HOL functions are deterministic and terminating, the programs we are able to model are also inherently so. We do not consider the termination requirement to be a severe restriction, but many, if not most, PETs involve probabilistic non-determinism in their design which is essential to their privacy guarantees. Fortunately, this restriction is easily overcome. Drawing again from Malacaria et al. [21], we introduce a component of the program input accounting for random behavior. This portion of the input state essentially serves as an oracle determining the resolution of the randomness on a given run of the program, with the distribution over the random behavior simply determined by the distribution over the random input states. For example, the following probabilistic algorithm involving a coin flip and a high and low input variable `if heads then l:=h else l:=l+1` becomes a deterministic algorithm involving high, low, and random input variables `if r == 1 then l:= h else l:=l+1`, assuming the *a priori* distribution on `r` is s.t. the probability that `r == 1` is $1/2$.

Proceeding from above, we define another HOL type β *prog_state*, which is the 4-tuple, $: \beta state * \beta state * \beta state * \beta state$, representing a possible program execution: (*high, low, random, output*).

Definition 9. A **program space** is a pair $(M, (s, \mu))$ of a HOL function modeling a program, $M(\text{high}, \text{low}, \text{random}) : (\beta state * \beta state * \beta state) \rightarrow \beta state$, and a probability space, (s, μ) , s.t. $s = \text{UNIV} : \beta \text{ prog_state}$ and $\forall h\ l\ r\ o. (o \neq M(h, l, r)) \Rightarrow (\mu(h, l, r, o) = 0)$.

The first condition in the definition of a program space, $s = \text{UNIV}$, ensures that any program execution (i.e. anything of type $: \text{prog_state}$) is measurable by μ ; the second condition ensures that only *valid* executions (i.e. those whose output is equal to the program applied to the inputs) have non-zero probability. The probability distribution μ over possible program executions for a particular program M captures the *a priori* distribution of high, low, and random input states and the *a posteriori* distribution on the outputs of M for these inputs. The four random variables

$$\begin{aligned} H &= \lambda x. \{(h, l, r, o) \mid h = x\}, & L &= \lambda x. \{(h, l, r, o) \mid l = x\}, \\ R &= \lambda x. \{(h, l, r, o) \mid r = x\}, & O &= \lambda x. \{(h, l, r, o) \mid o = x\} \end{aligned}$$

define equivalence relations on program executions based on the high input, low input, random input, and output respectively. In our HOL formalization, the random variables H , L , and O correspond to the random variables \mathcal{H} , \mathcal{L} , and \mathcal{O} in the definition of *leakage* for deterministic programs above. (Note: H , L , and O are random variables from \mathbf{s} to \mathbf{s} , so `random_variable_prob (s,mu) H h` reduces to `mu(H h)` and similarly for L and O and `random_vector_prob`). Now we can formalize leakage in HOL for a deterministic program M and a program space $(M, (\mathbf{s}, \mu))$ as

Definition 10 (Leakage for deterministic programs).

$$\text{leakage } (M, (\mathbf{s}, \mu)) = \text{conditional_mutual_information } (\mathbf{s}, \mu) \ O \ H \ L.$$

We have successfully formalized leakage for a deterministic program in HOL, but before defining information leakage for a probabilistic program, we must consider our view of the probabilistic non-determinism in the program. There are two possible views of the non-determinism in the program corresponding to differing capabilities of an observer/attacker. On the one hand, we have the case where an attacker, who is trying to learn about the high input from the output, can observe (but not control) the outcome of the random events (e.g. knows that the coin-flips came up heads) in a particular execution of the program. We shall term this view of the probabilistic behavior as *visible* probabilism, since the resolution of the non-determinism is known to the observer. On the other hand, we have the case where the attacker knows that the resolution of the nondeterminism follows a particular distribution (e.g. knows that a fair coin is flipped), but cannot observe directly how the nondeterminism is resolved in a particular execution of the program (e.g. doesn't know that the coin-flip was heads this time). We shall term this view of the probabilistic behavior as *hidden* probabilism, since the resolution of the non-determinism is hidden from the observer.

If we consider a program to have visible probabilism, then the leakage of the program is the amount of information about the high-security inputs that can be learned from the outputs given knowledge of the low-security inputs *and* the resolution of the non-determinism; however if we consider a program to have hidden probabilism, then the leakage of the program is the amount of information about the high-security inputs that can be learned from the outputs given knowledge of the low-security inputs *only* (i.e. without knowledge of the resolution of the non-determinism). This understanding of leakage is captured by defining leakage to be $\mathbb{I}(\mathcal{O}; \mathcal{H} | (\mathcal{L}, \mathcal{R}))$, for programs with visible probabilism, and $\mathbb{I}(\mathcal{O}; \mathcal{H} | \mathcal{L})$, for programs with hidden probabilism, where \mathcal{R} is a random variable ranging over the *random* inputs to the program that resolve the nondeterminism in a particular run of the program. We can now formalize information leakage of a nondeterministic program M and a program space $(M, (\mathbf{s}, \mu))$ as

Definition 11 (Leakage for programs with visible probabilism).

$$\text{leakage } (M, (\mathbf{s}, \mu)) = \text{conditional_mutual_information } (\mathbf{s}, \mu) \ O \ H \ (L++R),$$

Definition 12 (Leakage for programs with hidden probabilism).

$$\text{leakage } (\mathbb{M}, (\mathbf{s}, \mu)) = \text{conditional_mutual_information } (\mathbf{s}, \mu) \text{ } \mathbb{O} \text{ } \mathbb{H} \text{ } \mathbb{L}.$$

Note: we could analyze a program that contains *both* visible and hidden probabilism by splitting the random input into a visible component and hidden component and conditioning the measurement of conditional mutual information on the *visible* component of the random input.

It is important to distinguish these two views of the nondeterminism in our analysis of information leakage in order to be clear what we are measuring. The case study in Section 3 is an excellent example of this as there is no leakage if the probabilism is hidden (i.e. the outcomes of the coin flips are not known to an observer), but the leakage is total if the probabilism is visible (i.e. the outcomes of the coin flips are known to an observer). In [21], Malacaria et al. do not consider this distinction between the two views of nondeterminism in their analysis. They note that for deterministic programs $\mathbb{H}(\mathcal{O}|\mathcal{H}, \mathcal{L}) = 0$, so $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L}) = \mathbb{H}(\mathcal{O}|\mathcal{L})$. Malacaria et al. recognized this potential computational optimization for deterministic programs, but also noted that it may overestimate leakage for programs with probabilistic non-determinism [21]. Malacaria’s example is the program `l := random(0,1)`, which sets the output to be 0 or 1 with equal probability. For the program above $\mathbb{H}(\mathcal{O}|\mathcal{L}) = 1$, while $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L}) = 0$, so there is an overestimation of the leakage of the program. Malacaria proposes that this “extra” leakage can be attributed to uncertainty in the output resulting from the randomness of the program and can be eliminated by conditioning on the random input. Thus they define leakage for nondeterministic programs to be $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L}, \mathcal{R}) = \mathbb{H}(\mathcal{O}|\mathcal{L}, \mathcal{R})$, which is our definition of leakage for programs with *visible* nondeterminism only. Therefore, if we were to apply the definition for the leakage of nondeterministic programs adopted in [21] to the case study in Section 3, we would get the unintuitive result that a program which is known to be secure has a total leakage of information. We consider this distinction of visible and hidden probabilism in our definitions for information leakage of nondeterministic programs to be an important contribution of our work.

2.3 Assistance for the Uniformly Distributed Case

In order to prove some property of the information leakage of a program in HOL, we must first model the program and the probability space of possible program executions as HOL terms \mathbb{M} and (\mathbf{s}, μ) . Since our definitions for leakage in HOL are well defined for any \mathbb{M} and (\mathbf{s}, μ) of the appropriate type, we are obligated to prove that the (\mathbf{s}, μ) is in fact a probability space and that $(\mathbb{M}, (\mathbf{s}, \mu))$ is a program space. Recognizing that some effort is required for these proofs, we have defined a HOL function `unif_prog_space` which takes as its arguments a program modeled in HOL, \mathbb{M} , a set of high input states, `high`, a set of low input states, `low`, and a set of random input states, `random`; `unif_prog_space`(\mathbb{M} , `high`, `low`, `random`) is the program space for \mathbb{M} whose probability distribution is uniformly distributed over $\{(\mathbf{h}, \mathbf{l}, \mathbf{r}, \mathbb{M}(\mathbf{h}, \mathbf{l}, \mathbf{r})) \mid \mathbf{h} \in \text{high} \wedge$

$\{1 \text{ IN } \text{low} \wedge r \text{ IN } \text{random}\}$. We have proved that, for any M , high , low , and random which are finite and nonempty, `unif_prog_space` defines a valid program space. While we will not always be interested in analysis when the inputs are evenly distributed this will often be the case (as it is for the case study), so some initial effort has been eliminated for further applications of our technique. Furthermore, we have developed automation in HOL that proves the leakage of a `unif_prog_space` for small, finite instances nearly fully-automatically. This automation is only useful for small examples because its memory and computational overheads grow rapidly with the size of the example, but is still of value for automatically proving the base case of a general inductive proof of leakage or as an initial check of our intuitions about the leakage of a program.

3 Case Study: the Dining Cryptographers

Having outlined a method for proving the information leakage of a program in HOL, we now go on to demonstrate the applicability of this technique to proving privacy properties of PETs by proving the total-anonymity of the Dining Cryptographers protocol [4]. We begin below by developing a model for the Dining Cryptographers protocol in HOL using the techniques of the previous section; the protocol is briefly explained for the benefit of the unfamiliar reader. Finally, we discuss the relationship between information-leakage and anonymity for the dining cryptographers protocol. Our information-theoretic proof of anonymity for the dining cryptographers protocol in HOL4 is outlined in Appendix A.

3.1 Modeling the Dining Cryptographers Protocol in HOL

In Chaum’s original presentation of the Dining Cryptographers problem [4], a group of cryptographers sit down to dinner and are immediately informed by the Maître d’hôtel that the bill has already been paid. They come to the conclusion that either one of their party has paid, or the bill has been paid by some external agency such as the NSA. They would like to determine which of these has occurred while preserving the anonymity of the payer, in the event that one of the cryptographers has paid. The following solution is suggested. Each cryptographer flips a fair coin under the table and shares the outcome of the coin flip with the cryptographer to his left. All the cryptographers then announce the bitwise exclusive-or (xor) of the two coins they have seen (their own and that of the cryptographer to the right) and whether or not he/she has paid. If the bitwise xor of these announcements is true, then one of the cryptographers has paid, otherwise some outside agency has paid. The anonymity of the Dining Cryptographers protocol relies on the fairness of the coins and properties of xor; Chaum proved the total anonymity guarantee of the Dining Cryptographers protocol in [4]. Note that the nondeterminism inherent in the coin-flipping of the Dining Cryptographers protocol is that which we classified as *hidden* probabilism in the previous section, because the coins are flipped under the table where they cannot be observed.

We begin formalizing the Dining Cryptographers protocol in HOL by defining the sets of valid high-security, low-security, and random input-states. The type of state used is $: \text{string} \rightarrow \text{boolean}$ and since the state is infinite (there are an infinite number of possible variable names) we fix on the convention that any variable names we don't use map to the value \perp . The high input identifies which cryptographer has paid when one of them has and we index the cryptographers from 0 to $n - 1$, where there are n cryptographers. If the NSA has paid, the input is that n has paid. We can then define the set of valid high-security inputs as $\text{dc_high_states } n \top = \{(\lambda s. s = \text{"pays } n")\}$, when the NSA has payed, and $\text{dc_high_states } n \perp = \{(\lambda s. s = \text{"pays } i") \mid i < n\}$, when one of the cryptographers has paid. There are no low-security input variables, so we use $\text{dc_low_states} = \{(\lambda s. \perp)\}$ as our null low-input. The random inputs consist of all the possible combinations of coin flips for the n cryptographers, namely those states that map any variable other than "coin 0", ..., "coin $n - 1$ " to \perp :

$$\text{dc_random_states } n = \{x \mid \forall s. (\forall i. i \not< n \vee s \neq \text{"coin } i") \Rightarrow \neg x s\}.$$

We will now define the HOL function formalizing the Dining Cryptographers protocol; this is done in several parts corresponding to the various stages of the protocol. First the cryptographers' high-security coin values are set from the random-input. While this step is not strictly necessary and the coins from the random input could be used in the program directly (without affecting the analysis), this definition seems to reflect more directly the original definition. Note that HOL functions are typically defined recursively.

```

set_coins high random 0 =
  (\s. if s = "coin 0" then random s else high s)
set_coins high random (n + 1) =
  (\s. if s = "coin (n + 1)" then random s
    else set_coins high random n s)

```

The next step of the protocol is to set the announcements of each cryptographer. Since our cryptographers are indexed linearly, $0, \dots, n - 1$, rather than sitting around a circular table, we adopt the convention that for $0 \leq i < n - 1$ cryptographer $i + 1$ looks at coin $i + 1$ and coin i and cryptographer 0 looks at coin 0 and coin $n - 1$.

```

set_announcements low high n 0 =
  (\s. if s = "announces 0" then
    (high "pays 0") xor (high "coin 0") xor (high "coin n")
    else high s)
set_announcements low high n (i + 1) =
  (\s. if s = "announces i + 1" then
    (high "pays i + 1") xor (high "coin i + 1") xor (high "coin i")
    else set_announcements low high n i s)

```

The final step of the protocol is to determine the result (whether one of the cryptographers has paid or not) by xor-ing their announcements. This is done by defining two HOL functions, a helper-function to compute the bitwise xor of the announcements and the function that sets the result to this value.

```
xor_announces low 0 = low "announces 0"
xor_announces low (i + 1) =
  (low "announces i + 1") xor (xor_announces low i)

compute_result low n =
  (λs. if s = "result" then xor_announces low n else low s)
```

All that is necessary to complete our formalization of the Dining Cryptographers protocol in HOL is the definition of the function that connects the stages we have defined.

```
dcprog (n + 3) high low random =
  compute_result(set_announcements low
    (set_coins high random (n + 2))
    (n + 2) (n + 2))
  (n + 2)
```

Note: the function defining the overall protocol is for the argument $n + 3$, where $0 \leq n$, because the protocol is only valid for three or more cryptographers. The program space modeling the Dining Cryptographers protocol is the one in which the probability distribution is evenly distributed over the valid input-states identified above; this can easily be defined using `unif_prog_space` described in the previous section as

```
dc_prog_space n nsapays =
  unif_prog_space (dc_prog n)
  (dc_high_states nsapays n) dc_low_states (dc_random_states n).
```

3.2 An Information-Theoretic Proof of the Dining Cryptographers

Having formalized a program space modeling the Dining Cryptographers protocol above, we can analyze the information leakage of this protocol using the information-theoretic technique developed in the previous section. We are interested in how much information might be leaked by the protocol about the identity of a cryptographer who has payed the bill, so we focus on the case where one of the cryptographers has payed; the proof of correctness when the NSA has payed is relatively straightforward. We recall from above that leakage measures how many bits of the high-security input can be learned by someone who knows the low-security inputs and observes the outputs; depending on whether or not

we deem the probabilism in the protocol to be visible or hidden, the observer may also know the resolution of the probabilistic behavior in the protocol. In the case of the Dining Cryptographers protocol, there are no low-security inputs to be known and we consider the probabilism to be hidden, since the coins are flipped under the table. The outputs of the protocol are the announcements each cryptographer makes and the overall result (i.e. the xor of the announcements). The high-security input states whether or not cryptographer i has paid, for each of the $0 \leq i \leq n - 1$ cryptographers. Exactly one of the cryptographers will have paid, so there are $n - 1$ possible high inputs which are equally likely *a priori*. The high-security input is precisely the identity of the cryptographer who has paid and amounts to $\lg(n - 1)$ bits of information. In terms of our anonymity analysis this means that a leakage measure of $\lg(n - 1)$ would denote that *all* of the bits of the high-security input can be learned, so an observer can positively identify the payer and anonymity is completely compromised. If we were to obtain a leakage measurement of $(\lg(n - 1))/2$, then *half* of the bits of the high-security input can be learned and an observer can eliminate half of the cryptographers as candidates for being the payer. A leakage measurement of 0 would mean that an observer cannot learn anything about the identity of the payer from the outputs and total anonymity is maintained.

A proof of the total anonymity of the Dining Cryptographers protocol in HOL using the formalizations developed above can be found in Appendix A. The goal we prove is that, for three or more cryptographers ($n \geq 3$), when one of the cryptographers has paid (`nsapays = ⊥`), the information leakage of `dc_prog_space` is 0. Due to space constraints, we are only able to outline the HOL proof; the full script for the proof in HOL4 is approximately 1700 lines. When outlining the proof we choose to focus on the more interesting information-leakage aspects of the proof rather than the routine correctness aspects of the proof. Recall that every step of the proof is guaranteed by HOL4 to be logically consistent with our formalization of the protocol, probability theory, and any other mathematical theories used.

4 Summary and Future Work

Above we have developed an information-theoretic technique for proving the information leakage of programs formalized in the HOL4 theorem-prover. The advantage of this method being information-theoretic is that it is quantitative and capable of capturing *partial* information-leakage. Furthermore, by formalizing our analysis in HOL4, any proofs about information-leakage are guaranteed to be logically and mathematically consistent (up to the level of abstraction of the formalization).

After explaining this technique for proving information-leakage in HOL, we demonstrated its applicability to proving privacy properties of PETs by proving the total-anonymity of the Dining Cryptographers protocol [4]. As far as the author is aware, this is the first proof of the anonymity of the Dining Cryptographers protocol, for an *unbounded* number of cryptographers, mechanized in a

theorem-prover using a quantitative, information-theoretic metric for anonymity. Since it was first proposed, Chaum’s Dining Cryptographers protocol has frequently been used as an initial case study for many formal methods for analyzing privacy [1, 8, 25]. Now twenty years after it was first proposed, the Dining Cryptographers may be considered a “toy” example when compared to deployed PETs, but it remains useful as an initial proof-of-concept for analysis techniques before applying them to more realistic examples.

As the author considers the case study above as a proof-of-concept for the methodology developed in this paper, future work will include the use of this technique to prove privacy properties of deployed PETs. There are numerous possible applications including anonymous communications systems such as [7, 13, 15, 19, 23, 24]. Another interesting use would be to analyze the privacy leakage of algorithms designed to “anonymize” databases e.g. for releasing anonymized versions of government-controlled medical-databases for research purposes.

Another area for future work, is to develop a more robust attacker model for this analysis technique. As presented above, the attacker is a passive observer who cannot inject arbitrary messages into the system, etc. This is sufficient to model many attacks for the dining cryptographers e.g. two cryptographers sitting across from each other can collude and determine which side of the table the payer is on (We model this by making the coins the two cryptographers see part of the *visible* probabilism). However, a passive attacker is insufficient for most attacks on more sophisticated examples, so this remains an important future development for this line of research.

4.1 Acknowledgments

The author would like to thank Larry Paulson, Mike Gordon, Joe Hurd, and Magnus Myreen for numerous useful discussions related to this work, the Gates Cambridge Trust for funding this research, and the anonymous reviewers for their many helpful comments.

References

- [1] Mohit Bhargava and Catuscia Palamidessi. *Probabilistic Anonymity*, volume 3653. 2005.
- [2] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Symposium on Security and Privacy*, pages 140–154, May 2006.
- [3] Konstantinos Chatzikokolakis. *Probabilistic and Information-Theoretic Approaches to Anonymity*. PhD thesis, Laboratoire d’Informatique (LIX), École Polytechnique, Paris, October 2007.
- [4] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.
- [5] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. *Electr. Notes Theor. Comput. Sci.*, 59(3), 2001.
- [6] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *J. Log. Comput.*, 15(2):181–199, 2005.

- [7] G. Danezis, R. Dingleline, and N. Mathewson. Mixminion: design of a type III anonymous remailer protocol. pages 2–15, 2003.
- [8] Yuxin Deng, Catuscia Palamidessi, and Jun Pang. Weak probabilistic anonymity. In *Proceedings of SECCO '05*, Electronic Notes in Theoretical Computer Science, 2005.
- [9] Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.
- [10] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
- [11] A. K. Dewdney. Computer recreations: Of worms, viruses, and core war. *Scientific American*, page 110, March 1989.
- [12] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. *Towards Measuring Anonymity*, volume 2482. 2003.
- [13] Roger Dingleline, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
- [14] J. L. Doob. *Measure Theory*. Number 143 in Graduate Texts in Mathematics. Springer, 1991.
- [15] Sharad Goel, Mark Robson, Milo Polte, and Emin Gun Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. Technical Report 2003-1890, Cornell University, Ithaca, NY, February 2003.
- [16] Michael J. C. Gordon. From lcf to hol: a short history. In G. Plotkin, Colin P. Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction*. MIT Press, 2000.
- [17] O. Hasan and S. Tahar. Verification of expectation properties for discrete random variables in hol. *Theorem Proving in Higher-Order Logics*, (4732):119–134, September 2007.
- [18] Joe Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, 2002.
- [19] Brian Neil Levine and Clay Shields. Hordes — A Multicast Based Protocol for Anonymity. *Journal of Computer Security*, 10(3):213–240, 2002.
- [20] Gavin Lowe. Breaking and fixing the needham-schroder public-key protocol using fdr. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.
- [21] Pasquale Malacaria. Assessing security threats of looping constructs. In *POPL*, pages 225–235, 2007.
- [22] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.
- [23] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. Technical Report 97-15, DIMACS, 1997.
- [24] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002.
- [25] Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *ESORICS*, pages 198–218, 1996.
- [26] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In Roger Dingleline and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

- [27] Claude E. Shannon. A mathematical theory of communication. *Bell System Technincal Journal*, (27):379–423 and 623–656, July and October 1948.
- [28] Vitaly Shmatikov. Probabilistic model checking of an anonymity system. In Steve Schneider, editor, *Journal of Computer Security*, volume 12(3/4), pages 355–377, 2004.
- [29] David Williams. *Probability with Martingales*. Cambridge Mathematical Textbooks. Cambridge University Press, 1991.

A Proof of anonymity for the Dining Cryptographers

We define `program`, `prob`, and `events` s.t. `program` takes a program space $(M, (s, \mu))$ as its argument and gives the program component M . Similarly, `prob` and `events` give the probability measure and the events of a program space respectively. We abbreviate

$p\ n = \text{prob } (\text{dc_prog_space } (n+3) \perp)$, $e\ n = \text{events } (\text{dc_prog_space } (n+3) \perp)$,
 $\text{prog } n = \text{program } (\text{dc_prog_space } (n+3) \perp)$,
 $\text{valid } n = \{s \mid \text{prob } n\ s \neq 0\}$, $\text{high } n = \text{dc_high_states } (n+3) \perp$,
(similarly for `dc_low_states` and `dc_random_states`), and finally
 $\text{output } n = \text{IMAGE } (\lambda x. \text{prog } n\ x) (\text{high } n\ \text{CROSS } \text{low } n\ \text{CROSS } \text{random } n)$.

Lemma 1 (There are $(n+3)(2^{n+3})$ valid program-states).

$$\forall n. \text{CARD } (\text{valid } n) = (n+3)(2^{n+3})$$

Proof. Follows directly from basic definitions. Intuitively, there are $n+3$ possible high-inputs, 2^{n+3} possible random-inputs, and 1 low-input.

Lemma 2 (A valid program-state has probability $1/((n+3)(2^{n+3}))$).

$$\forall n\ s. s \text{ IN } (\text{valid } n) \Rightarrow (p\ n\ \{s\} = 1/((n+3)(2^{n+3})))$$

Proof. Follows directly from basic definitions and Lemma 1. This result is obvious, since the valid program-states are uniformly distributed.

Lemma 3 (There are 2 valid states for a given output and high-input).

$$\forall o\ h\ n. o \text{ IN } \text{output } n \wedge h \text{ IN } \text{high } n \Rightarrow \\ (\text{CARD}\{r \mid (h, (\lambda s. \perp), r, o) \text{ IN } \text{valid } n\} = 2)$$

Proof. We begin by defining a function which determines if a random input r is valid for output o when cryptographer h pays:

$$\begin{aligned} \text{coins_valid } r\ o\ h\ n\ 0 = & \\ & (r\ \text{“coin0”} = r\ \text{“coin}(n+2)\text{”} \text{ xor } (\text{XOR } o\ 0) \text{ xor } (0 \not\leftarrow h)) \\ \text{coins_valid } r\ o\ h\ n\ (i+1) = & \\ & ((r\ \text{“coin}(i+1)\text{”} = & \\ & \quad r\ \text{“coin}(n+2)\text{”} \text{ xor } (\text{XOR } o\ (i+1)) \text{ xor } (i+1 \not\leftarrow h)) \wedge \\ & \text{coins_valid } r\ o\ h\ n\ i), \end{aligned}$$

where $\text{XOR out } i$ is the pointwise xor of the announcements 0 through i . It is then straightforward to prove

$$\begin{aligned} \forall n \ h \ o. \ h \leq (n+2) \ \wedge \ o \ \text{IN output } n \Rightarrow \\ (\{r \mid ((\lambda s. s = \text{"pays } h"), (\lambda s. \perp), r, o) \ \text{IN valid } n\} = \\ \{r \mid r \ \text{IN random } n \ \wedge \ \text{coins_valid } r \ o \ h \ n \ (n+2)\}) \end{aligned} \quad (1)$$

by induction. We then define a function that constructs a valid random-input given o , h , and our choice of whether the first coin is heads or tails

$$\begin{aligned} \text{make_r } o \ h \ n \ \text{choice } 0 &= (\lambda s. \text{ if } s = \text{"coin0"} \ \text{then} \\ &\quad \text{choice xor (XOR out 0) xor (0 } \not\prec h) \ \text{else } \perp) \\ \text{make_r } o \ h \ n \ \text{choice } (i+1) &= (\lambda s. \text{ if } s = \text{"coin}(i+1)" \ \text{then} \\ &\quad \text{choice xor (XOR out } (i+1)) \ \text{xor } (i+1 \not\prec h) \ \text{else} \\ &\quad \text{make_r } o \ h \ n \ \text{choice } i \ s) \end{aligned}$$

and prove by induction that

$$\begin{aligned} \forall n \ h \ o. \ h \leq (n+2) \ \wedge \ o \ \text{IN output } n \Rightarrow \\ (\{r \mid r \ \text{IN random } n \ \wedge \ \text{coins_valid } r \ o \ h \ n \ (n+2)\} = \\ \{\text{make_r } o \ h \ n \ \top \ (n+2); \text{make_r } o \ h \ n \ \perp \ (n+2)\}) \end{aligned} \quad (2)$$

from which our goal immediately follows.

Lemma 4 (There are $2(n+3)$ valid states for a given output).

$$\forall n \ o. \ o \ \text{IN output } n \Rightarrow (\text{CARD } \{s \mid s \ \text{IN valid } n \ \wedge \ s \ \text{IN } 0 \ o\} = 2(n+3))$$

Proof. By the definition of valid and set operations, it is sufficient to prove that

$$\begin{aligned} \forall n \ o. \ o \ \text{IN output } n \Rightarrow & \left(\text{CARD } (\text{IMAGE } (\lambda (h, r). (h, (\lambda s. \perp), r, o)) \right. \\ & \left. \{(h, r) \mid h \ \text{IN high } n \ \wedge \ (h, (\lambda s. \perp), r, o) \ \text{IN valid } n\} \right) \\ & = 2(n+3) \end{aligned}$$

and by a property of the cardinality of the image of an injective function

$$\begin{aligned} \forall n \ o. \ o \ \text{IN output } n \Rightarrow \\ (\text{CARD } \{(h, r) \mid h \ \text{IN high } n \ \wedge \ (h, (\lambda s. \perp), r, o) \ \text{IN valid } n\} = 2(n+3)). \end{aligned}$$

Our goal follows from Lemma 3, and that there $n+3$ possible high-states.

Lemma 5 (The probability of a valid output is $1/2^{n+2}$).

$$\forall n \ o. \ o \ \text{IN output } n \Rightarrow (p \ n \ 0 \ o = 1/2^{n+2})$$

Proof. By Lemma 2 and basic definitions the goal is equivalent to

$$\forall n \ o. \ o \text{ IN output } n \Rightarrow \\ (\text{SUM } (\lambda s. 1/((n+3)(2^{n+3})))\{s \mid s \text{ IN valid } n \wedge s \text{ IN } 0 \ o\}) = 1/2^{n+2}.$$

Since the above is a summation over a constant function, it is equivalent to

$$\forall n \ o. \ o \text{ IN output } n \Rightarrow \\ \left((1/((n+3)(2^{n+3})))(\text{CARD}\{s \mid s \text{ IN valid } n \wedge s \text{ IN } 0 \ o\}) = 1/2^{n+2} \right),$$

which is easily provable by Lemma 4 and basic arithmetic.

Lemma 6 (There are 2^{n+2} valid outputs).

$$\forall n. \text{CARD}(\text{output } n) = 2^{n+2}$$

Proof. Similarly to Lem. 3, we begin by defining a function that constructs the list of valid outputs,

$$\begin{aligned} \text{announces_list } 0 &= [(\lambda s. s = \text{“announces0”}); (\lambda s. \perp)] \\ \text{announces_list } (i+1) &= \\ &\left(\text{MAP } (\lambda s. (\lambda x. \text{if } x = \text{“announces}(i+1)\text{” then } \top \text{ else } s \ x)) \right. \\ &\quad \left. \text{announces_list } i \right) ++ \\ &\left(\text{MAP } (\lambda s. (\lambda x. \text{if } x = \text{“announces}(i+1)\text{” then } \perp \text{ else } s \ x)) \right. \\ &\quad \left. \text{announces_list } i \right) \\ \text{outputs_list } (n+3) &= \text{MAP } (\lambda l \ s. (s = \text{“result”}) \vee \\ &\quad (\text{if } s = \text{“annouces}(n+2)\text{” then } \neg(\text{XOR } l \ (n+1)) \text{ else } l \ s)) \\ &\quad \text{announces_list } (n+1). \end{aligned}$$

Building on proofs of the correctness of our model of the d.c. protocol, we can inductively prove that $\forall n \ x. \ x \text{ IN output } n = x \text{ MEM outputs_list } (n+3)$. Two relatively straightforward inductive proofs yield that

$$\forall n. \text{ALL_DISTINCT}(\text{outputs_list } n) \tag{3}$$

(all the members of `outputs_list` are distinct) and that

$$\forall n. \text{LENGTH}(\text{outputs_list } (n+3)) = 2^{n+2} \tag{4}$$

from which our goal follows.

Theorem 1 (Conditional entropy of 0 given L is $n+2$).

$$\forall n. \text{conditional_entropy } (e \ n, p \ n) [0] [L] = n+2$$

Proof. Since there is only one valid low-input, $(\lambda s.\perp)$, the goal reduces to

$$\forall n. - \text{SUM } (\lambda o. (\text{p } n \ 0 \ o)(\text{lg}(\text{p } n \ 0 \ o))) (\text{output } n) = n + 2$$

by Definitions 4 and 6 and basic arithmetic. Using Lemma 5 the goal becomes

$$\forall n. ((1/2^{n+2})(\text{lg}(1/2^{n+2}))) (\text{CARD } (\text{output } n)) = n + 2$$

which follows from Lemma 6, properties of lg , and basic arithmetic.

Lemma 7 (Probability of a high state is $1/(n+3)$).

$$\forall n \ h. h \text{ IN high } n \Rightarrow (\text{p } n \ H \ h = 1/(n+3))$$

Proof. Follows from the fact that there are $n+3$ evenly distributed high-inputs.

Lemma 8 (There are 2 states with a given output and high-input).

$$\forall n \ h \ o. h \text{ IN high } n \wedge o \text{ IN IMAGE } (\lambda r. \text{prog } n \ (h, (\lambda s.\perp), r)) \text{ random } n \Rightarrow \\ (\text{CARD}\{s \mid s \text{ IN H } h \wedge s \text{ IN O } o \wedge s \text{ IN valid } n\} = 2)$$

Proof. By the basic definitions for the protocol, the goal is equivalent to

$$\forall n \ h \ o. h \leq (n+2) \wedge o \text{ IN IMAGE } (\lambda r. \text{prog } n \ ((\lambda s.s = \text{"pays } h"), (\lambda s.\perp), r)) \\ \text{ random } n \Rightarrow (\text{CARD}\{r \mid ((\lambda s.s = \text{"pays } h"), (\lambda s.\perp), r, o) \text{ IN valid } n\} = 2),$$

which follows from (1) and (2) in Lemma 3.

Lemma 9 (Probability of a valid output & high input is $1/((n+3)2^{n+2})$).

$$\forall n \ h \ o. h \text{ IN high } n \wedge o \text{ IN IMAGE } (\lambda r. \text{prog } n \ (h, (\lambda s.\perp), r)) \text{ random } n \Rightarrow \\ (\text{random_vector_prob } (e \ n, p \ n) \ [0; H] \ [o; h] = 1/((n+3)2^{n+2}))$$

Proof. By Lemma 2 and basic definitions, the goal becomes

$$\forall n \ h \ o. h \text{ IN high } n \wedge o \text{ IN IMAGE } (\lambda r. \text{prog } n \ (h, (\lambda s.\perp), r)) \text{ random } n \Rightarrow \\ (\text{SUM } (\lambda s. 1/((n+3)2^{n+3})) \\ \{s \mid s \text{ IN O } o \wedge s \text{ IN H } h \wedge s \text{ IN valid } n\} = 1/((n+3)2^{n+2})),$$

which further reduces to

$$\forall n \ h \ o. h \text{ IN high } n \wedge o \text{ IN IMAGE } (\lambda r. \text{prog } n \ (h, (\lambda s.\perp), r)) \text{ random } n \Rightarrow \\ ((1/((n+3)2^{n+3}))(\text{CARD}\{s \mid s \text{ IN O } o \wedge s \text{ IN H } h \wedge s \text{ IN valid } n\}) = \\ 1/((n+3)2^{n+2}))$$

by properties of SUM . The above follows from Lem. 8 and basic arithmetic.

Lemma 10 (There are 2^{n+2} valid outputs for a given high-input).

$$\forall n \ h. h \text{ IN high } n \Rightarrow (\text{CARD}(\text{IMAGE } (\lambda r. \text{prog } n \ (h, (\lambda s.\perp), r)) \text{ random } n) = 2^{n+2})$$

Proof. By (3) and (4) of Lemma 6 it is sufficient to prove that

$$\forall n \ h \ x. \ h \ \text{IN} \ \text{high } n \Rightarrow \left(x \ \text{IN} \ \text{IMAGE} \ (\lambda r. \ \text{prog } n \ (h, (\lambda s. \perp), r)) \right. \\ \left. \text{random } n = x \ \text{MEM} \ \text{outputs_list} \ (n + 3) \right)$$

which can be proved using the correctness properties of our definition of the protocol. Further details of this proof are omitted due to space constraints.

Lemma 11 (Conditioned entropy of 0 given H and L is $-(n + 2)$).

$$\forall n \ h. \ h \ \text{IN} \ \text{high } n \Rightarrow \\ \left(\text{conditioned_entropy} \ (e \ n, p \ n) \ [0] \ [H; L] \ [h; (\lambda s. \perp)] = -(n + 2) \right)$$

Proof. Since there is only one valid low-input, $(\lambda s. \perp)$, the goal reduces to

$$\forall n \ h. \ h \ \text{IN} \ \text{high } n \Rightarrow \\ \left(- \text{SUM} \ (\lambda o. \ (\lambda x. \ x(\lg x)) \ (\text{random_vector_prob} \ (e \ n, p \ n) \ [0; H] \ [o; h]) / \right. \\ \left. (p \ n \ H \ \{h\})) \ \text{IMAGE} \ (\lambda r. \ \text{prog } n \ (h, (\lambda s. \perp), r)) \ \text{random } n = -(n + 2) \right)$$

by Def. 6, which then further reduces to

$$\forall n \ h. \ h \ \text{IN} \ \text{high } n \Rightarrow \\ \left(- \left(1/2^{n+2} (\lg(1/2^{n+2})) \right) \right. \\ \left. (\text{CARD}(\text{IMAGE} \ (\lambda r. \ \text{prog } n \ (h, (\lambda s. \perp), r)) \ \text{random } n) = -(n + 2) \right)$$

by Lems. 7 and 9 and properties of SUM. The above follows from Lem. 10, a property of the cardinality of the image of an injective function, and basic arithmetic.

Theorem 2 (Conditional entropy of 0 given H and L is $n + 2$).

$$\forall n. \ \text{conditional_entropy} \ (e \ n, p \ n) \ [0] \ [H; L] = n + 2$$

Proof. Since there is only one valid low-input, $(\lambda s. \perp)$, the goal reduces to

$$\forall n. \ - \text{SUM} \ (\lambda h. \ (p \ n \ H \ h) (\text{conditioned_entropy} \ (e \ n, p \ n) \ [0] \ [H; L] \ [h; (\lambda s. \perp)])) \\ \text{high } n = n + 2$$

which is equivalent to

$$\forall n. \ \text{SUM} \ (\lambda h. \ (n + 2) / (n + 3)) \ \text{high } n = n + 2$$

by Lemmas 7 and 11 and basic arithmetic. Since the summation above is of a constant function, the goal reduces to

$$\forall n. \ (\text{CARD}(\text{high } n)) ((n + 2) / (n + 3)) = n + 2,$$

which follows directly from basic arithmetic and the fact that there are $n + 3$ valid high-inputs.

Theorem 3 (The Dining Cryptographer's protocol preserves anonymity).

$$\forall n. \ \text{leakage} \ (\text{dc_prog_space} \ (n + 3) \ \perp) = 0$$

Proof. Follows directly from Defs. 8 and 12, Thms. 1 and 2, and basic arithmetic.