Information Hiding, Anonymity and Privacy: A Modular Approach

Dominic Hughes *
STANFORD UNIVERSITY

Vitaly Shmatikov †
SRI INTERNATIONAL

Abstract

We propose a new specification framework for information hiding properties such as anonymity and privacy. The framework is based on the concept of a *function view*, which is a concise representation of the attacker's partial knowledge about a function. We describe system behavior as a set of functions, and formalize different information hiding properties in terms of views of these functions. We present an extensive case study, in which we use the function view framework to systematically classify and rigorously define a rich domain of identity-related properties, and to demonstrate that privacy and anonymity are independent.

The key feature of our approach is its modularity. It yields precise, formal specifications of information hiding properties for *any* protocol formalism and *any* choice of the attacker model as long as the latter induce an observational equivalence relation on protocol instances. In particular, specifications based on function views are suitable for any cryptographic process calculus that defines some form of indistinguishability between processes. Our definitions of information hiding properties take into account any feature of the security model, including probabilities, random number generation, timing, *etc.*, to the extent that it is accounted for by the formalism in which the system is specified.

Keywords: security, information hiding, logic, knowledge, Kripke structure, verification, anonymity, privacy

1 Introduction

Security requirements for computer systems often involve hiding information from an outside observer. Secrecy, anonymity, privacy, and non-interference each require that it should be impossible or infeasible to infer the value of a particular system attribute — a transmitted credit card number, the identity of a website visitor, the sender and recipient of an email message — by observing the system within the constraints of a given

*Computer Science Department, Stanford University, Stanford, CA 94305 U.S.A. Email address: dominic@cs.stanford.edu.

†SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 U.S.A. Email address: shmat@csl.sri.com. Partially supported by ONR grants N00014-02-1-0109 and N00014-01-1-0837 and DARPA contract N66001-00-C-8015.

observer model. If formal analysis techniques are to be used for the analysis and verification of computer security, they must provide support for the formal specification of information hiding properties as well as formal reasoning about information leaked to the attacker by various events and actions occurring in the system.

In this paper we adopt an epistemological tradition that can be traced back to the seminal works of Kripke [Kri63] and Hintikka [Hin62]: hiding information, modeled as the attacker's lack of knowledge about the system, corresponds to indistinguishability of system states. As the starting point, we assume that we are given a set of system configurations $\mathbb{C}$ equipped with an observational equivalence relation $\sim$. Consequently, our methods apply to *any* computational model of the attacker that partitions the space of all possible system configurations into observational equivalence classes. A typical example is the specification of a security protocol in a cryptographic process calculus whose notion of equivalence $\sim$ is testing equivalence of processes in some attacker model.

The following informal example illustrates the way in which we shall obtain formal definitions of security properties, parametrically in $\sim$. For ease of presentation, in this example we restrict to the case where a communication or exchange between agents consists of a single message (for example, an email). Thus we have in mind a Kripke structure whose possible worlds (states) are all possible email exchanges, and for which $C \sim C'$ represents the attacker's inability to distinguish between possible worlds C and C'. Below is a natural-language expansion of the predicate we shall later obtain (in Table 2) for defining absolute sender anonymity:

ABSOLUTE SENDER ANONYMITY holds if:
for every possible world C of the Kripke structure,
  for every message m sent in C,
    for every agent a,
      there exists a possible world C' indistinguishable from C (*i.e.*, $C' \sim C$)
      such that in C', a is the sender of m.

Thus, from the attacker's perspective, the lineup of candidates for the sender of any given message is the entire set of agents. (More generally, m would denote a full exchange or 'conversation' between agents, potentially consisting of more than one message, and transmitted through any medium.) This example, though informal, should convey the idea behind our formulation of security properties parametrically in $\sim$.

A key advantage of this modularity with respect to $\sim$ is the resulting leveraging of the expressive power of the underlying formalism (*e.g.*, process calculus) in which a protocol is specified. Depending on the formalism, the $\sim$ equivalence relation, which represents the attacker's inability to distinguish certain system states, may take into account features such as probability distributions, generation of nonces and new names, timing, etc. In this case, our framework and the formal specifications of information hiding properties derived using it will also take these features into account.

Our framework. Information hiding properties can be formalized naturally using modal logics of knowledge [FHMV95, SS99]. Such logics can be used to formulate direct statements about the limits of an observer's knowledge. Verifying whether a system satisfies a given information hiding property is more difficult, since in order to reason about information flows, it is necessary to formalize the behavior of all agents comprising the system as "knowledge-based programs." This is often a non-trivial exercise, requiring expertise in the chosen logic.

On the other end of the formal methods spectrum, approaches based on process algebras [Sch96, AG99, LMMS99, BNP99] are well suited to formalizing concurrent systems. A process algebra may include a formal model of cryptographic primitives — needed, *e.g.*, for the analysis of cryptographic protocols — and typically comes equipped with an equivalence relation such as bisimulation or testing equivalence that models an observer's inability to distinguish between certain processes. Process algebras also provide proof techniques for process equivalence. The disadvantage of the process algebra approach is that stating information hiding properties in terms of process equivalence is very subtle and error-prone, especially for complicated properties such as anonymity and privacy.

We introduce a modular framework for reasoning about information hiding properties, independent of any particular system specification formalism or epistemic logic (see Figure 1). The cornerstone of our approach is the concept of a *function view*. A function view is a foundational domain-theoretic notion. It represents partial information about a function, and thus models an observer's incomplete knowledge thereof. Remarkably, just the three attributes of a function view — graph, kernel, and image — suffice to model many kinds of partial knowledge an observer may have about the function of interest. We demonstrate how any system specification formalism that provides an equivalence relation on system configurations induces function views, and how information hiding properties can be stated naturally in terms of *opaqueness* of this view. Therefore, security properties specified in our framework may be amenable for formal verification using a wide variety of formal methods and techniques.

Applications to anonymity and privacy. The application of our specification framework to anonymity and privacy is especially significant. Identity protection is an active area of computer security research. Many systems have been proposed that implement different, and sometimes even contradictory, notions of what it means to be "anonymous." Instead of a single "anonymity" or "privacy" property, there are dozens of different flavors of anonymity, and understanding them in a systematic way is a major challenge in itself. There is also a need for rigorous formal specification and verification of identity-related security properties since such properties are often difficult to model using conventional formal analysis techniques.

Structure of paper. The structure of the paper is as follows. In section 2, we introduce function views. In section 3, we show how opaqueness of function views can be used to formalize information hiding properties. In section 4, we use our theory of function views and opaqueness to demonstrate how most notions of anonymity proposed in the research literature can be formalized in a uniform way and represented as predicates

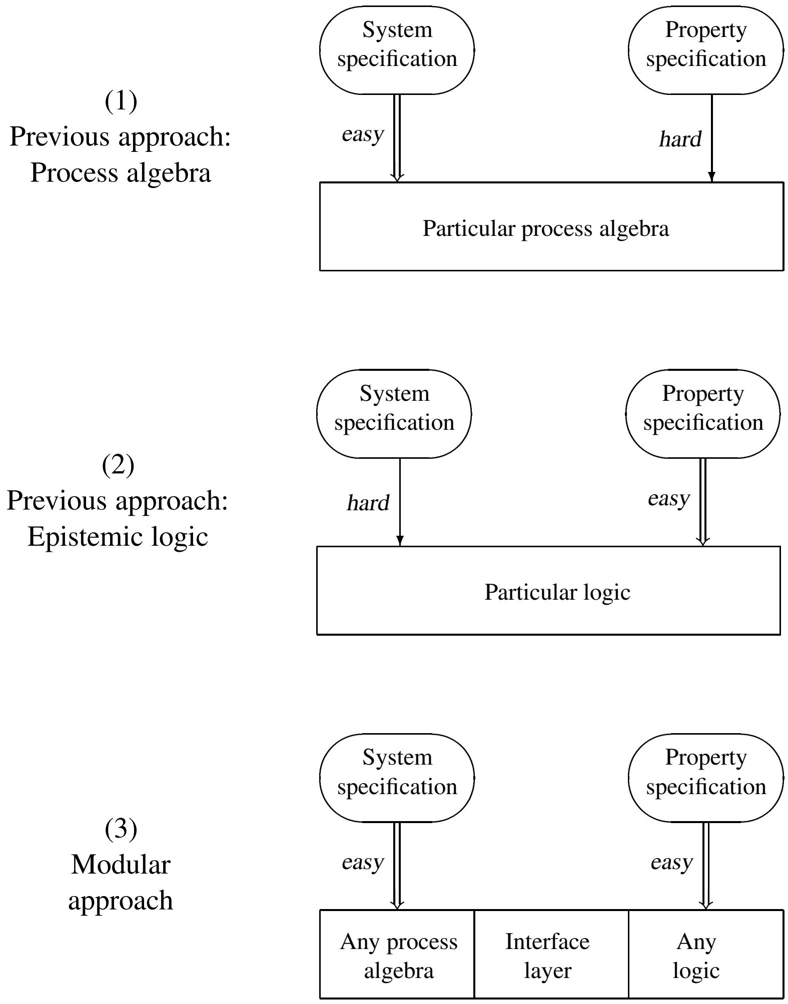

Figure 1: Modular approach to formalizing information hiding properties. See section 3.6 for a detailed explanation.

on observational equivalence classes, thus facilitating their verification in any cryptographic process algebra. Perhaps our most important practical result is a crisp distinction between anonymity and privacy (the latter understood as "relationship anonymity"), which has implications for public policy.

2 Theory of Function Views

What can one *know* about a function $f : X \to Y$? One might know its output $f(x)$ on a certain input x, or that a certain point y of Y lies in the image of f but that another point y' does not. One may know distinct inputs x and x' such that $f(x) = f(x')$, without necessarily knowing the value $f(x)$.

One approach to modeling partial knowledge of a function is to use domain theoretic ideas [Sco72, AJ94], defining an approximation a of a function $f : X \to Y$ to be any partial function $a \subseteq f\ (\subseteq X \times Y)$. This traditional notion of an approximation as a subset of input-output behavior has been very successful in research into semantics of programming languages [Sto77, Gun92].

In this paper we introduce a new notion of partial knowledge of a function. A *view* of a function f comprises a non-deterministic approximation of its graph (a binary relation containing f), a subset of its image, and an equivalence relation contained in its kernel. Function views form a distributive lattice whose maximal consistent elements correspond to fully determined functions and whose bottom element represents absence of any knowledge. In section 2.1 we define three primitive forms of *opaqueness*, one for each of component of a view, each formalizing an observer's inability to discern certain information about f.

In section 3 we show how to formalize information hiding properties in terms of opaqueness of functions defining system behavior. The most important aspect of this formalization is that *any* Kripke structure representing an observer gives rise to function views. In sections 3.2 and 3.4, we show how function views are constructed automatically from the equivalence relation of a Kripke structure. We then show how any opaqueness property can be formalized as a predicate on equivalence relations, hence on Kripke structures.

In particular, if the system is specified in a process algebra that supports a notion of observational equivalence $\sim$, we demonstrate how opaqueness-based security properties of the system can be expressed as predicates on $\sim$. This conversion is parametric in $\sim$ in the sense that it works for any computational model of the attacker that partitions the space of all possible system configurations into observational equivalence classes. Therefore, we do not require technical machinery for reasoning about how the partial knowledge represented by function views is obtained. This knowledge is implicit in the equivalence relation induced by the attacker model. Since opaqueness properties can be expressed as predicates on the equivalence classes of *any* process algebra, a user is free to employ his or her favorite algebra and preferred technique for verifying such predicates.

Our chosen definition of function view is simple yet expressive enough to formalize a host of security properties and the relationships between them, as shown in section 4. It is interesting and surprising that attacker knowledge in this complex setting can be reduced to triples consisting of an approximation of the graph, image and kernel of a function. In section 2.5, we discuss the scope of our framework, including its limitations and possible extensions.

2.1 Partial knowledge of functions

We re-examine our original question: what can one *know* about a function? Consider the following properties of a function $f : X \to Y$.

- Its ***graph.*** The input-output behavior of f, typically coded as a subset of $X \times Y$.
- Its ***image.*** The subset $\mathsf{im}\, f = \{f(x) \mid x \in X\}$ of Y.
- Its ***kernel.*** The quotient induced by f, *i.e.*, the equivalence relation $\mathsf{ker}\, f$ on X given by $\langle x, x' \rangle \in \mathsf{ker}\, f$ *iff* $f(x) = f(x')$.

This list is by no means exhaustive. These three properties suffice, however, for formalizing many information hiding properties such as anonymity and privacy, and we focus on them for the rest of this paper. We define the following corresponding notions of knowledge of a function $f : X \to Y$.

- ***Graph knowledge.*** A binary relation $F \subseteq X \times Y$ such that $f \subseteq F$. Thus F is a non-deterministic approximation of f: $F(x) = \{ y \in Y \mid xFy \}$ is a set of candidates for the output of f on x, and $f(x)$ is always a candidate.
- ***Image knowledge.*** A subset I of Y such that $I \subseteq \mathsf{im}\, f$. The fact that $y \in I$ is an assertion that y is an output of f, without necessarily knowing any specific input that produces y.
- ***Kernel knowledge.*** An equivalence relation K on X such that $K \subseteq \mathsf{ker}\, f$, *i.e.*, xKx' only if $f(x) = f(x')$. Thus the edges of K are assertions about the equality of f.

Note that the second and third forms of knowledge are *positive* in the sense that each point $y \in I$ or $\langle x, x' \rangle \in K$ is a definite observation of the image or kernel. By contrast, graph knowledge F is *negative*, since $y \notin F(x)$ is a definite observation that $f(x) \neq y$, whereas $y \in F(x)$ does not imply $f(x) = y$ (unless $F(x)$ is a singleton).

DEFINITION 1 ***Function knowledge*** *of type $X \to Y$ is a triple $N = \langle F, I, K \rangle$ where*

1. *$F \subseteq X \times Y$ is a binary relation between X and Y,*
2. *I is a subset of Y, and*
3. *K is an equivalence relation on X.*

We say that $N = \langle F, I, K \rangle$ is ***consistent*** with f, denoted $N \sqsubseteq f$, if

1. $F \supseteq f$,
2. $I \subseteq \mathsf{im}\, f$, and
3. $K \subseteq \mathsf{ker}\, f$.

From a theoretical or foundational perspective, the choice of these three particular components is somewhat *ad hoc*. The interesting and surprising point is that these three components alone allowed us to express a rich spectrum of security-related properties: see Figure 2, and the taxonomy in section 4.3.

A ***lineup*** of type $X \to Y$ is a set of functions of type $X \to Y$. Given function knowledge N of type $X \to Y$, define the associated lineup $\mathsf{lin}(N) \subseteq Y^X$ as the set of functions with which N is consistent:

$$\mathsf{lin}(N) \;=\; \{\, f : X \to Y \mid N \sqsubseteq f \,\}$$

Under the intuition that N is an observer's view of some unknown function f, $\mathsf{lin}(N)$ is the set of candidates for f.

Given function knowledge $N = \langle F, I, K\rangle$ and $N' = \langle F', I', K'\rangle$ of type $X \to Y$, define $N \sqsubseteq N'$ (N ***approximates*** N', or N' ***refines*** N), by

$$N \sqsubseteq N' \quad\Longleftrightarrow\quad \begin{cases} F & \supseteq & F' \\ I & \subseteq & I' \\ K & \subseteq & K' \end{cases}$$

(conjunction on the right). Upon identifying $f : X \to Y$ with function knowledge $\langle f, \mathsf{im}\, f, \mathsf{ker}\, f\rangle$ of type $X \to Y$, this extends our earlier definition of consistency. Meet and join are pointwise:

$$\begin{aligned} \textstyle\bigwedge_{j\in J} \langle F_j, I_j, K_j\rangle &= \langle\, \textstyle\bigcup_{j\in J} F_j, \bigcap_{j\in J} I_j, \bigcap_{j\in J} K_j \,\rangle \\ \textstyle\bigvee_{j\in J} \langle F_j, I_j, K_j\rangle &= \langle\, \textstyle\bigcap_{j\in J} F_j, \bigcup_{j\in J} I_j, \bigvee_{j\in J} K_j \,\rangle \end{aligned}$$

where $\bigvee_{j\in J} K_j$ is the transitive closure of $\cup_{j\in J} K_j$. With top $\langle \emptyset, Y, X \times X\rangle$ and bottom $\langle X \times Y, \emptyset, =_X\rangle$, function knowledge of type $X \to Y$ thus forms a distributive lattice.

2.2 Knowledge closure

Our motivation for the notion of function knowledge $N = \langle F, I, K\rangle$ is to have a way to represent the knowledge of an observer who is trying to discern some properties of a hidden function f. Ideally, we would use N to formulate assertions about the observer's inability to discern certain properties of f. For example, if the cardinality of $F(x)$ is at least two elements for each $x \in X$, we can assert that "the observer cannot determine the value of f on any input."

In general, however, we cannot make direct assertions about information hiding in terms of a single component of the knowledge triple because the three components are not independent. For example, in the case above, suppose that K is the whole of $X \times X$

(so the observer knows that f produces the same output value on all inputs), and that I is the singleton $\{y\}$. Then from I and K the observer can infer that $f(x) = y$ for all x, even if the first coordinate F is such that $F(x)$ is always of size at least 2. In other words, from I and K, the observer can refine his or her knowledge F of the graph of f. In order to make sound assertions about information hiding using N, we must first take an appropriate form of deductive or inference closure. The correct definition arises as the closure operator induced by a Galois connection.

Given a lineup $l \subseteq Y^X$ of functions from X to Y, define the associated function knowledge $\mathsf{kn}(l)$ of type $X \to Y$ by $\mathsf{kn}(l) = \bigwedge l$. Thus $\mathsf{kn}(l)$ is the $\sqsubseteq$-maximal function knowledge consistent with each $f \in l$. (In making this definition we identify a function f with function knowledge $\langle f, \mathsf{im}\, f, \mathsf{ker}\, f\rangle$.) Therefore if $\mathsf{kn}(l) = \langle F, I, K\rangle$, then $F = \bigcup l$, $I = \bigcap\{\mathsf{im}\, f : f \in l\}$ and $K = \bigcap\{\mathsf{ker}\, f : f \in l\}$.

PROPOSITION 1 (GALOIS CONNECTION) *The maps $N \mapsto \mathsf{lin}(N)$ and $l \mapsto \mathsf{kn}(l)$ constitute a contravariant Galois connection between the lattice of function knowledge of type $X \to Y$ and the powerset-lattice of lineups of type $X \to Y$,* i.e., *$l \subseteq \mathsf{lin}(N)$ iff $N \sqsubseteq \mathsf{kn}(l)$.*

Proof. We must show that

$$l \quad \subseteq \quad \{\, f : X \to Y \mid N \sqsubseteq f \,\} \tag{1}$$

iff

$$N \quad \sqsubseteq \quad \bigwedge l \tag{2}$$

Suppose (1) holds. Then each $f \in l$ satisfies $N \sqsubseteq f$, so (2) follows immediately from the fact that $\bigwedge l$ is a meet.

Conversely, suppose (2) holds. Writing $N = \langle F, I, K\rangle$, (2) is equivalent to the conjunction of

$$F \quad \supseteq \quad \bigcup l \tag{3}$$

$$I \quad \subseteq \quad \bigcap\{\mathsf{im}\, f : f \in l\} \tag{4}$$

$$K \quad \subseteq \quad \bigcap\{\mathsf{ker}\, f : f \in l\} \tag{5}$$

We must show that for each $f \in l$, we have $N \sqsubseteq f$, *i.e.*, $F \supseteq f$, $I \subseteq \mathsf{im}\, f$, and $K \subseteq \mathsf{ker}\, f$. These follow from (3), (4) and (5) respectively. $\square$

Given function knowledge N define the ***closure*** $\overline{N}$ of N to be $\mathsf{kn}(\mathsf{lin}(N))$. Motivated by the discussion at the start of the section, henceforth we shall only deal with closed function knowledge. Note that there is no easy equational characterization of closure, especially in applications that involve additional components in the definition of function knowledge.

2.3 Function views and opaqueness

DEFINITION 2 *A **view** of a function $f : X \to Y$ is any closed function knowledge of type $X \to Y$ that is consistent with f.*

To formalize information hiding, we shall require the following predicates on views.

DEFINITION 3 (OPAQUENESS) *Let $V = \langle F, I, K \rangle$ be a view of $f : X \to Y$. We define the following forms of **opaqueness** of f under V:*

- ***Value opaqueness:***
  - *Given $n \geq 2$, V is n-**value opaque** if $|F(x)| \geq n$ for all $x \in X$. In other words, the lack of information in the view is such that there are at least n candidates for the output of f on any given input x.*
  - *Given $Z \subseteq Y$, V is Z-**value opaque** if $Z \subseteq F(x)$ for all $x \in X$. In other words, the lack of information in the view is such that any element of Z is a possibility for the value of f on any input x.*
  - *V is **absolutely value opaque** if V is Y-value opaque. In other words, the view gives away nothing about the input-output behavior of f: for all $x \in X$, every $y \in Y$ is a candidate for $f(x)$.*
- *V is **image opaque** if I is empty. In other words, the view is such that for no element $y \in Y$ can one definitely assert that y lies in the image of f.*
- *V is **kernel opaque** if K is equality on X. In other words, beyond the trivial case $x = x'$, with this view of f no equalities $f(x) = f(x')$ can be inferred.*

PROPOSITION 2 *Let f be a function of type $X \to Y$. The various forms of opaqueness of a view V of f can be characterized logically as follows, with reference to the corresponding lineup $l = \mathsf{lin}(V)$:*

$$\begin{aligned}
Z\text{-Value opaqueness} &\Leftrightarrow \forall x \in X.\, \forall z \in Z.\, \exists g \in l.\, g(x) = z. \\
n\text{-Value opaqueness} &\Leftrightarrow \forall x \in X.\, \exists g_1, \ldots, g_n \in l.\;\; i \neq j \;\Rightarrow\; g_i(x) \neq g_j(x) \\
\text{Image opaqueness} &\Leftrightarrow \forall y \in Y.\, \exists g \in l.\, y \notin \mathsf{im}\, g \\
\text{Kernel opaqueness} &\Leftrightarrow \forall x_1 \neq x_2 \in X.\, \exists g \in l.\, g(x_1) \neq g(x_2)
\end{aligned}$$

For absolute value opaqueness, take $Z = Y$.

Proof. Each property is simply a restatement of the corresponding property defined in Definition 3. $\square$

2.4 Composition of function views

Compositionality is a notoriously difficult problem in computer security. It is well known that security properties typically do not compose, and reasoning about what an observer can learn upon putting together two views of the system is very involved. Although composition of function views is of no particular interest from the perspective of security property specification (we assume that *all* information available in the attacker is already embodied in the indistinguishability relation that induces his function view), for the sake of completeness and mathematical curiosity, we remark upon it.

The only sensible definition of composition of function views is in terms of the underlying lineups: $v \cdot w = \mathsf{kn}(\{ f \cdot g : f \in \mathsf{lin}(v),\ g \in \mathsf{lin}(w) \})$. One must take closure, for in general the pairwise composite of closed lineups does not yield a closed lineup. Since in composing at type $A \to B \to C$, image and kernel information in B is lost, it is not surprising that composition is not associative, as witnessed by the following example. Let $2 = \{0, 1\}$, $3 = \{0, 1, 2\}$, and $R = \{\langle 0, 0\rangle, \langle 1, 0\rangle, \langle 2, 1\rangle\} \subset 3 \times 2$. Then for $u = \langle R, 2, = \cup\{\langle 0, 1\rangle, \langle 1, 0\rangle\}\rangle : 3 \to 2$, $v = \langle R^{-1}, \{2\}, =\rangle : 2 \to 3$ and $w = \langle 2 \times 2, \{0\}, =\rangle : 2 \to 2$, the constant function $\lambda x.1 : 2 \to 2$ is in $\mathsf{lin}(u \cdot (v \cdot w))$ but is not in $\mathsf{lin}((u \cdot v) \cdot w)$, hence $u \cdot (v \cdot w) \neq (u \cdot v) \cdot w$.

Since image and kernel information in B is lost upon composing $A \to B \to C$, the only route to a compositional notion of function view would be to somehow encode this information elsewhere, either by enriching the image or kernel components, or by adding one or more additional components. For example, the above non-associativity of the composition of u, v and w arises because in closing $w \cdot v$ we lost the information that 0 is in the image of w. The fact that $0 \in \mathsf{im}\, w$ implies the following condition (C): at least one of 0 and 1 is in the image of $w \cdot v$. Since C is lost, the closure of $w \cdot v$ is "too big," including as it does the constant function $\lambda x.2$, which does not satisfy C. Motivated by this example, one avenue towards a compositional enrichment of the notion of function view might begin with extending the definition of the image component from a subset of the range to a *set* of subsets of the range, with the semantics that each such set contains at least one point of the image. Then the lost condition C above can be encoded as the set $\{0, 1\}$, and we eliminate the undesirable function $\lambda x.2$ from the closure of $w \cdot v$. A similar extension may be necessary for the kernel.

2.5 Scope of the framework

Depending on the application, one might consider incorporating additional properties, for example the cardinality of $f^{-1}(y)$ for each y, or a subset of the complement of the kernel. Because of our description of "inference closure" between components of function knowledge as a Galois connection with function lineups (Proposition 1), the theory of function views extends to *any* conceivable property of a function. In addition, one can easily generalize from views of functions to views of partial functions or relations.

As we apply the function view framework to reasoning about information hiding in systems, we assume that the equivalence relation associated with the observer is non-probabilistic. For example, we do not consider scenarios where the observer does not know with 100% certainty the value of $f(x)$, but may be able to determine that

$f(x) = y_1$ with probability 90%, and $f(x) = y_2$ with probability 10%. While it restricts applicability of the framework, this approach is common in formal reasoning about security properties. It is justified by assuming that cryptographic primitives hide distributions of the underlying data, and it is therefore sufficient to consider only non-probabilistic observers such as the so called *Dolev-Yao* attacker [DY83]. Any notion of process equivalence with sound cryptographic semantics [AR02, LMMS99] provides a non-probabilistic equivalence relation suitable for applying the function view framework. Nonetheless, future extensions may consider probability distributions over function (and hence attribute) lineups in a manner similar to Halpern and Tuttle [HT93]. In the current setup, a lineup of type $X \to Y$ is equivalent to a function $Y^X \to \{0, 1\}$. This has an obvious generalization to $Y^X \to [0, 1]$, where $[0, 1]$ is the closed unit interval on the real line.

Our theory of function views does not directly incorporate any temporal features. We are primarily concerned with reasoning about information that an observer may extract from the system given a particular static equivalence relation that models his or her inability to distinguish certain system configurations. As the system evolves over time, the observer may accumulate more observations, possibly narrowing the equivalence relation and allowing extraction of more information. This can be modeled in our framework by the corresponding change in the observer's function view.

If temporal inferences, such as those involved in timing attacks, can be modeled in the underlying process specification formalism, they will be reflected in the induced observational equivalence relation and, in a modular fashion, in the function views representing the attacker's view of the system. Therefore, function views can be used to reason about time-related security properties. Function views can also be used to represent partial knowledge about relations between unknown entities. Therefore, they are sufficiently expressive to model "forward security" of systems. For example, kernel opaqueness of the mapping from email messages to senders models the attacker's inability to determine whether two emails originated from the same source. Even if one of the messages is compromised, the attacker will not be able to automatically infer the sender of the other.

3 Opaqueness and Information Hiding

In this section, we establish the relationship between opaqueness of function views and observational equivalence of system configurations. We then demonstrate how function views can be used to formalize information hiding properties and derive verification conditions stated in terms of observational equivalence.

3.1 Possible-worlds model

We will use the theory of function views developed in section 2.1 to reason about information hiding properties of systems, *i.e.*, whether the attacker is prevented from knowing properties of the functions defining system behavior. We will follow the standard approach of epistemic logic [Hin62, FHMV95] and formalize any system of interest S

as a *Kripke structure* $\langle \mathbb{C}, \pi, \mathcal{K} \rangle$ [Kri63]. Here $\mathbb{C}$ is the set of all possible *configurations* of system S (we may also refer to elements of $\mathbb{C}$ as *possible worlds* or *states* of S), π is an interpretation that defines configurations $C \in \mathbb{C}$ by assigning values to all attributes of S, and $\mathcal{K}$ is an equivalence relation on $\mathbb{C}$ that models an observer's inability to distinguish certain states of C. In general, Kripke structures can be used to model multiple observers with their corresponding equivalence relations on $\mathbb{C}$. Since our primary goal in this paper is to reason about security properties, we will assume a single attacker that incorporates the abilities of all hostile observers.

In the rest of this section, we demonstrate how any Kripke structure induces function views. In particular, any computational model of the attacker, including those implicit in cryptographic process algebras, imposes an equivalence relation on $\mathbb{C}$ and thus induces function views. Any information hiding property can be defined in two ways: as opaqueness of the induced function view (following section 2.3), or as a logical predicate on the underlying equivalence relation. The two definitions are equivalent, as demonstrated by proposition 3.

3.2 Attribute opaqueness

Let S be a system with a set of configurations $\mathbb{C}$. An ***attribute*** α of S of type $X \to Y$ is a function $\alpha_C : X \to Y$ for each configuration $C \in \mathbb{C}$, *i.e.*, a $\mathbb{C}$-indexed family of functions $\langle \alpha_C : X \to Y \rangle_{C \in \mathbb{C}}$. In general, S may have a variety of attributes. Such a representation is akin to the object-oriented view of the world, with behavior modeled by a set of methods.

Security properties of computer systems often involve hiding information about functions defining the behavior of the system. For example, suppose S is a bank with a set of customers X. Writing customer x's bank balance in configuration $C \in \mathbb{C}$ as $\mathsf{bal}_C(x)$, we have defined an attribute bal of S of type $X \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers. Then the secrecy property of customers' balances can be formalized as the requirement that an observer should not be able to infer the value of attribute bal. A richer example is noninterference [GM82], which requires that the observable *low* (unclassified) behavior of the system hide all information about *high* (classified) functions inside the system.

Define a ***view family*** V for an attribute α of S to be a function view V_C of $\alpha_C : X \to Y$ for each $C \in \mathbb{C}$. Opaqueness lifts pointwise to attributes as follows.

DEFINITION 4 *Let S be a system with set of configurations $\mathbb{C}$ and an attribute α of type $X \to Y$. Let V be a view family for α. For any form of opaqueness ϕ (*e.g., ϕ=kernel *or ϕ=*image*), we say that α **is ϕ-opaque under** V if, for all configurations $C \in \mathbb{C}$, $\alpha_C : X \to Y$ is ϕ-opaque under the function view V_C.*

3.3 Observational equivalence

Intuitively, two processes are observationally equivalent if no context can distinguish them. In formal models of security protocols based on process calculi [AG99, LMMS99, BNP99], it is common to define security properties such as secrecy and authentication in terms of observational equivalence (or related concepts such as may-testing equivalence or computational indistinguishability) between several instances of the protocol, or an instance of the protocol and that of an "ideal" protocol which is secure by design and serves as the specification of the desired property. Proof techniques for observational equivalence depend on the choice of a particular process calculus and an attacker model. Some of the formalisms are amenable to mechanized verification (*e.g.*, via trace-by-trace comparison). A related approach involves behavioral equivalences proved via logical relations [SP01].

Given system configurations P and Q, we will say that $P \sim_E Q$ *iff* an outside observer (attacker), acting in coalition with all agents from set E (*e.g.*, with access to their secret keys), cannot distinguish P and Q, for whatever notion of indistinguishability is supported by the chosen formalism and the attacker model.

We emphasize that *any* notion of observational equivalence $\sim$ automatically provides a relation $\mathcal{K}$ for the Kripke structure defining the attacker's knowledge of the system. Therefore, once the system is formalized in a suitable process algebra, there is no need to reason how function views are obtained or where attacker knowledge "comes from." As demonstrated in section 3.4, any observational equivalence relation induces a particular function view, and its opaqueness can be characterized logically in terms of predicates on the equivalence classes as described in section 3.5.

3.4 Opaqueness and observational equivalence

Suppose that the set of system configurations $\mathbb{C}$ is equipped with an ***observational equivalence*** $\sim$, an equivalence relation where $C \sim C'$ represents the inability of an observer to distinguish between configurations C and C'. Such an equivalence relation naturally induces a view family for any attribute, as follows.

DEFINITION 5 *Let S be a system with set of configurations $\mathbb{C}$ equipped with an observational equivalence $\sim$, and let α be an attribute of S of type $X \to Y$. Every configuration $C \in \mathbb{C}$ defines a function lineup $L^{\sim}_{\alpha,C} = \{\, \alpha_{C'} : X \to Y \mid C' \sim C \,\}$, hence we obtain an attribute view family $\mathsf{view}^{\sim}(\alpha)$ of α, given by $\mathsf{view}^{\sim}(\alpha)_C = \mathsf{kn}(L^{\sim}_{\alpha,C})$*

Note that $\mathsf{view}^{\sim}(\alpha)_C$ is indeed a view of α_C because $\alpha_C \in L^{\sim}_{\alpha,C}$ and $\mathsf{kn}(l)$ is closed for any function lineup l. Since any observational equivalence induces an attribute view family, any form of opaqueness lifts to a predicate on observational equivalence.

DEFINITION 6 *Let S be a system with set of configurations $\mathbb{C}$ and let α be an attribute of S of type $X \to Y$. Let $\sim$ be an observational equivalence on $\mathbb{C}$. For any form of opaqueness ϕ, we say that α **is ϕ-opaque under** $\sim$ if the attribute view family $\mathsf{view}^{\sim}(\alpha)$ of α induced by $\sim$ is ϕ-opaque.*

3.5 Logical characterization of attribute opaqueness

Proposition 2 generalises to attribute opaqueness in the obvious way.

PROPOSITION 3 *Let S be a system with set of configurations $\mathbb{C}$ equipped with an observational equivalence $\sim$, and let α be an attribute of S of type $X \to Y$. Opaqueness of α under $\sim$ can be characterized as follows:*

$$\begin{array}{rcl}
Z\text{-Value opaqueness} & \Leftrightarrow & \forall C \in \mathbb{C}.\, \forall x \in X.\, \forall z \in Z.\, \exists C' \sim C.\, \alpha_{C'}(x) = z. \\
n\text{-Value opaqueness} & \Leftrightarrow & \forall C \in \mathbb{C}.\, \forall x \in X.\, \exists C_1, \dots, C_{n-1} \in \mathbb{C}.\, C_i \sim C \\
 & & \wedge\, \alpha_{C_i}(x) \neq \alpha_C(x) \,\wedge\, [i \neq j \Rightarrow \alpha_{C_i}(x) \neq \alpha_{C_j}(x)] \\
\text{Image opaqueness} & \Leftrightarrow & \forall C \in \mathbb{C}.\, \forall y \in Y.\, \exists C' \sim C.\, y \notin \mathsf{im}\, \alpha_{C'} \\
\text{Kernel opaqueness} & \Leftrightarrow & \forall C \in \mathbb{C}.\, \forall x_1 \neq x_2 \in X.\, \exists C' \sim C.\, \alpha_{C'}(x_1) \neq \alpha_{C'}(x_2)
\end{array}$$

For absolute value opaqueness, take $Z = Y$.

Proof. An immediate corollary of Proposition 2, since attribute opaqueness is defined pointwise. $\square$

3.6 Formalization of information hiding properties

We now present a modular two-step approach to formalizing information hiding properties, illustrated with a case study in section 4. The first step is to represent the system in question as a set of configurations parameterized by the relevant attributes. Given an intuitive notion of a security property, we specify the property in terms of attribute opaqueness (see Table 1 in the case study for examples).

The second step is to define an observational equivalence relation $\sim$ on configurations: $C \sim C'$ if and only if an observer cannot distinguish between C and C'. If the system is specified in a suitable cryptographic calculus, *e.g.*, [AG99, BNP99, SP01], the calculus will provide such a relation. Definition 6 then immediately yields a formal equivalence-based definition of the security property, which can be written as a logical predicate on $\sim$ following Proposition 3 (see Table 2 for examples).

Our method applies to any formalism that induces an equivalence relation on the space of all possible system configurations. In particular, it applies to any cryptographic process algebra formalism and any notion of process indistinguishability associated with the chosen attacker model. It permits formalization of security properties which would otherwise be difficult to model directly in process calculus. A key advantage of the approach is that our formalization of security properties is independent of the computational model of the observer, since it is based on function views. In many cases, the obtained observational equivalence properties can be verified using proof techniques of the underlying calculus.

Figure 1 summarizes the difference between conventional approaches to formalizing information hiding and the modular approach proposed in this paper. Any approach requires formalization of both the system (*e.g.*, a communication protocol) and the desired property, as represented by the downward arrows in each of the three sub-figures. Sub-figures (1) and (2) depict conventional approaches. In (1), a particular process algebra naturally models the system. However, reasoning about the knowledge an observer can extract from the system is fairly complicated (*e.g.*, [SS96]). Dually, in (2), a particular logic of knowledge is used to naturally model the knowledge available to an observer, but the system may no longer be specified in a standard process algebra (*e.g.*, [SS99]). Process algebras and epistemic logics are typically results of years of evolution specific to their respective domains of application. In an approach based on one of the two, the other is pushed beyond its natural boundaries.

Our goal is to leverage the best of both approaches, with an interface between the two. Sub-figure (3) depicts the modular approach introduced in this paper. Technically, we achieve modularity by combining our theory of function views (section 2) and the attacker's observational equivalence relation implicit in the process algebra (section 3.3). In fact, the interface layer is designed so that *any* process algebra or logic can be employed — in our framework, specification of information hiding properties is not linked to a particular choice of either. This may not always be sufficient to ensure that the property can be verified using the proof techniques of some algebra, since the specification is in terms of a predicate on the observational equivalence relation $\sim$, which may be difficult to verify directly. Nevertheless, it is a useful step towards bridging the gap between logic-based specification of desired information hiding properties and process algebra-based specification of system behavior.

4 Case Study: Anonymity and Privacy

Protection of personal information is one of the most important security goals of communication protocols on public networks. This goal can often be characterized as a set of secrecy properties that must be satisfied by the protocol. Properties that relate to the secrecy of data (credit card numbers, medical records, *etc.*) have been extensively researched in the literature on security protocol analysis. In this study, we are concerned with *anonymity* and *privacy*, that is, the secrecy of the identities of communicating agents as well as the secrecy of relationships between agents, such as the patient-doctor or sender-recipient relationship. Note that relationship secrecy is different from the conventional notion of data secrecy. For example, if an observer can find out that a particular person is a patient at an HIV clinic, this may violate privacy even if the actual contents of communication remain secret.

In sections 4.1 and 4.2, we formalize systems of communicating agents as protocol graphs. When modeling a specific system configuration, an edge in the protocol graph represents the fact that a certain pair of agents are engaged in a conversation, while abstracting away from the details of the actual data exchange (an edge is an abstraction for any number of messages that may flow between the communicating agents as part of the protocol). Protocol graphs can be used to model communication in any medium, including pairwise channels, broadcast, connectionless communication, *etc.*

Each protocol graph can be fully defined as a collection of functions, enabling us to apply the function view framework developed in section 2 and formally state information hiding properties as opaquenesses of certain function views. In section 4.3, we present a taxonomy of anonymity and privacy properties, followed by a related discussion in section 4.4. In section 4.5, we demonstrate how properties provided by existing anonymity systems can be expressed as opaquenesses of certain functions and, therefore, as predicates over the observational equivalence classes of the system. In section 4.6, we show how to define privacy as relationship anonymity, and in section 4.7 demonstrate that anonymity and privacy (expressed as relationship anonymity) are independent. Finally, in section 4.8 we show how the protocol graph model can be extended so that function views can be applied to model richer anonymity properties such as pseudonymity.

Related work. Syverson and Stubblebine proposed a special-purpose epistemic logic for reasoning about anonymity in [SS99]. In general, the main advantage of modal logics of knowledge is that even fairly complex information hiding properties can be stated directly as formulas in the logic. To verify whether a given system satisfies a particular anonymity property, it is necessary, however, to formalize the behavior of system agents as knowledge-based programs [FHMV95], which is non-trivial and requires expert knowledge of the chosen logic.

On the other end of the spectrum, Schneider and Sidiroupoulos [SS96] give a definition of anonymity using CSP process calculus, a well-understood process specification formalism. In general, using a process algebra greatly facilitates the task of formally specifying the behavior of system agents. Unfortunately, formally stating information hiding properties is quite difficult. The paper considers only one form of anonymity, and it is not immediately clear how it can be extended to cover other forms of anonymity and privacy such as untraceability or relationship anonymity.

Our modular approach allows us to develop formal definitions of a wide range of anonymity and privacy properties in a manner that does not depend on the choice of a particular formalism, and is therefore complementary to both modal logics such as that of [SS99] and process algebras such as that of [SS96]. Logical formulas describing the desired information hiding properties can be easily characterized as opaquenesses of certain function views, converted into predicates over observational equivalence classes, and verified, when possible, using the proof techniques of the chosen process formalism without the need to specify the system as a collection of knowledge-based programs. In fact, even CSP can be used as such a formalism. Our technique combines the benefits of the knowledge-based approach, namely, natural specification of information hiding properties, with those of the process algebra-based approach, namely, natural specification of system behavior. It is also sufficiently expressive to model many different flavors of anonymity and privacy, as demonstrated by section 4.3.

4.1 Protocol graphs

Let A be a finite set of ***agents*** $\{a, b, \dots\}$. Given a set T of ***relationship types***, let $\tau : A \times A \to T$ be an assignment of types to ordered pairs of agents. For example, in a protocol for mobile phone communications, we can let $T = \{\mathsf{subscriber}, \mathsf{roaming}, \mathsf{unauthorised}\}$, and interpret $\tau(ph149, GSMnet) = \mathsf{roaming}$ as "mobile phone $ph149$ has roaming rights on network $GSMnet$." In general, τ helps model protocols where a participant's behavior varies depending on the identity of the counterparty. A more general framework might express a dynamically changing τ, or even τ that is an arbitrary program.

Given any two-party communication protocol $\mathcal{P}$, let $\pi_{\mathcal{P}}(a, b, \tau, m)$ be the term in a suitable process calculus representing a single instance of $\mathcal{P}$ between agents a and b, parameterized by the relationship typing τ, and with m a unique identifier marking the instance. The process $\pi_{\mathcal{P}}(a, b, \tau, m)$ is parameterized by τ because, in general, the protocol executed between a and b will depend on the value of $\tau(a, b)$. For example, if b is the website of a medical clinic, and $T = \{\mathsf{patient}, \mathsf{nonpatient}\}$, then $\pi_{\mathcal{P}}(a, b, \tau, m)$ will depend on whether or not a is a patient of b.

Consider a concurrent execution of one or more protocol instances:

$$P = \pi_{\mathcal{P}}(a_1, b_1, \tau, m_1) \mid \cdots \mid \pi_{\mathcal{P}}(a_k, b_k, \tau, m_k) \mid S_{\mathcal{P}}$$

Each identifier or marker m_i is distinct since we make the standard assumption that the attacker can observe the communication medium and distinguish different instances of communication taking place, without necessarily knowing the identities of the communicating agents. $S_{\mathcal{P}}$ represents the unobservable system process, if any (*e.g.*, $S_{\mathcal{P}}$ may model the behavior of a secure anonymous remailer). We refer to each term $\pi_{\mathcal{P}}(a_i, b_i, \tau, m_i)$ as a ***conversation*** between a and b. Although unnatural from a linguistic point of view, we speak of a as the ***sender*** and b as the ***recipient*** of the conversation, since this makes our subsequent definitions of anonymity properties consistent with the standard terminology (*e.g.* [PKS01]). Sender a and recipient b are treated asymmetrically because, in general, the protocol specification may prescribe different behavior depending on whether a participant is playing the sender or the recipient role. Note that, in the process displayed above, we do not preclude the possibility that $a_i = a_j$ or $b_i = b_j$ for some $i \neq j$, or $a_i = b_j$ for some i and j. In other words, the same agent may be involved in several conversations, either as sender or recipient.

Any observable communication based on $\mathcal{P}$ can be represented by a protocol graph, defined below. First, we require two auxilary graph-theoretic definitions.

DEFINITION 7 *A* ***multigraph*** (V, E, s, t) *consists of a set* V *of vertices, a set* E *of edges, a source function* $s : E \to V$*, and a target function* $t : E \to V$*.*

There is an important distinction between multigraphs and classical graphs. In a classical graph, consisting of a vertex set V and set of edges $E \subseteq V \times V$, each edge *implicitly* determines its endpoints. By contrast, a multigraph is a set E of "abstract edges" and a set of V of vertices together with functions $s, t : E \to V$ assigning a source and target vertex to each edge. It is quite possible that two edges have the same source and target. This, for example, will allow for the possibility of two concurrent conversations between a clinic c and a patient p, modeled by two edges $e, e' \in E$ each with source $s(e) = s(e') = c$ and target $t(e) = t(e') = p$.

DEFINITION 8 *A* ***colored multigraph*** (G, K, k) *is a multigraph* $G = (V, E, s, t)$ *together with a set* K *(of colors) and an assignment* $k : V \times V \to K$ *of colors to pairs of vertices.*

Note that here we are coloring *ordered pairs* of vertices, and not vertices or edges, as is usual for colored graphs. This allows us to represent relationship types as colors (*e.g.*, clinic-patient, network-subscriber). Simply coloring vertices would not be sufficient. Suppose we have three vertices, a clinic c, a clinic c', and a person p. To express the fact that p is a patient of c, but not a patient of c', we need to color ordered pairs: $k(c,p) = \mathsf{patient}$ and $k(c',p) = \mathsf{non\text{-}patient}$.

For convenience and brevity, we shall denote a colored multigraph (G, K, k), with $G = (V, E, s, t)$, by the associated diagram of functions:

$$E \xrightarrow{\langle s,t\rangle} V \xrightarrow{k} K$$

Here, given functions $f : X \to Y$ and $g : X \to Z$, we use $\langle f, g\rangle$ to denote the canonically associated function of type $X \to Y \times Z$ (*i.e.*, $\langle f, g\rangle(x) = \langle f(x), g(x)\rangle \in Y \times Z$). Now we are ready for the main definition in this section.

DEFINITION 9 *A* ***protocol graph*** $C = (\mathsf{s}_C, \mathsf{r}_C, \tau_C)$ *over* M, A *and* T *is a colored multigraph with edge set* M, *vertex set* A, *and set of colors* T, *as follows:*

$$M \xrightarrow{\langle \mathsf{s}_C,\mathsf{r}_C\rangle} A \times A \xrightarrow{\tau_C} T$$

We write $\mathbb{C}_{M,A,T}$ *for the set of protocol graphs over* M, A *and* T.

We refer to the edges of M as ***abstract conversations***, the vertices of A as ***agents***, the colors of T as ***relationship types***, the source function s_C as the ***sender*** function, the target r_C as the ***recipient*** function, and τ_C as the ***relationship typing*** (or ***typing***) function. We shall write $\mathsf{sr}_C : M \to A \times A$ as shorthand for the ***sender-recipient*** function $\langle \mathsf{s}_C, \mathsf{r}_C\rangle : M \to A \times A$.

The communication medium is by assumption *observable*, *i.e.*, an observer can detect that some communication has occurred (*e.g.*, by overhearing an encrypted message on a broadcast network), even though its origin and destination may be unobservable. Therefore, each member of M is literally an "abstract conversation," *i.e.*, a conversation with unknown endpoints. A typical goal of the attacker is to find the mappings s_C and r_C from the conversations to their senders and recipients.

Depending on the protocol, the attacker may also be able to observe $\chi \stackrel{\mathsf{def}}{=} \tau_C \circ \mathsf{sr}_C$. For example, in a medical clinic example, if the protocols for patients and nonpatients are observably different, the attacker may able to determine whether a particular instance of communication with the clinic originates from a patient or a nonpatient, without necessarily knowing the originator's identity.

Given a two-party communication protocol $\mathcal{P}$, as described at the beginning of section 4.1, every observable communication

$$P = \pi_{\mathcal{P}}(a_1, b_1, \tau, m_1) \mid \cdots \mid \pi_{\mathcal{P}}(a_k, b_k, \tau, m_k) \mid S_{\mathcal{P}}$$

determines a protocol graph $C(P)$ in the obvious way, over

$$\begin{aligned} M &= \{ m_i \mid i = 1, \ldots, k \} \\ A &= \{ a_i \mid i = 1, \ldots, k \} \cup \{ b_i \mid i = 1, \ldots, k \} \end{aligned}$$

and T the set of relationship types specified as part of $\mathcal{P}$: define sender $\mathsf{s}_{P(C)}(m_i) = a_i$, recipient $\mathsf{r}_{P(C)}(m_i) = b_i$, and typing $\tau_{P(C)} = \tau$ (recall that τ, as part of the data of $\mathcal{P}$, is *a priori* a coloring of agent pairs). Naturally $P(C)$ is also a protocol graph over M, A' and T' for any $A' \supseteq A$ and $T' \supseteq T$.

Conversely, given a protocol graph C over M, A and T, the corresponding protocol instances can be reconstructed as

$$P(C) = \pi_{\mathcal{P}}(\mathsf{s}_C(m_1), \mathsf{r}_C(m_1), \tau_C) \mid \cdots \mid \pi_{\mathcal{P}}(\mathsf{s}_C(m_n), \mathsf{r}_C(m_n), \tau_C) \mid S_{\mathcal{P}}$$

where $m_i \in M$ are the abstract conversations (edges).

We emphasize that an edge in the protocol graph represents an entire *conversation* between two agents, not just a single message. The protocol graph model abstracts away from the details of the underlying communication medium. The presence of an edge in the graph with sender a and recipient b does *not* indicate that there is an established pairwise channel between agents a and b; it models only the fact that a and b are communicating, *i.e.*, are engaged in an instance of the communication protocol. The actual communication may be conducted (for example) via a broadcast medium, and may involve a sequence of messages, each of which may require generating new nonces, computing encryptions, *etc.* So long as the actual message exchange is modeled in a process calculus that gives rise to observational equivalence classes on system configurations, the function view framework from section 2 is applicable.

4.2 Observational equivalence of protocol graphs

Observational equivalence on processes induces observational equivalence on protocol graphs as follows. For protocol graphs C, C' over M, A and T (*i.e.*, $C, C' \in \mathbb{C}_{M,A,T}$), define ***graph equivalence***

$$C \overset{\mathcal{P}}{\sim} C' \quad \equiv \quad P(C) \sim_{A \setminus \mathsf{actv}(C,C')} P(C')$$

where $\mathsf{actv}(C, C')$ is the set of agents who communicated in C or C':

$$\mathsf{actv}(C, C') = \mathsf{s}_C(M) \cup \mathsf{r}_C(M) \cup \mathsf{s}_{C'}(M) \cup \mathsf{r}_{C'}(M)$$

The intuition behind the restriction $A \setminus \mathsf{actv}(C, C')$ is that an agent can always observe the difference between two instances of the protocol if, for example, he communicated in one instance but not the other. The instances (modeled as protocol graphs) are equivalent if any coalition of agents *not* involved in conversations in either instance cannot observe any difference.

4.3 Taxonomy of atomic anonymity and privacy properties

Given our abstract representation of observable communication as a protocol graph C over M, A and T, *i.e.*,

$$M \xrightarrow{\mathsf{sr}_C} A \times A \xrightarrow{\tau_C} T$$

we systematically obtain a space of anonymity and privacy properties by considering opaqueness of the following attributes (families of functions indexed by C):

$$\begin{aligned}
\mathsf{s} &: M \to A \\
\mathsf{r} &: M \to A \\
\mathsf{sr} &: M \to A \times A \\
\tau &: A \times A \to T \\
\tau^a &: A \to T \\
\tau^a \circ \mathsf{s} &: M \to T \\
\tau^a \circ \mathsf{r} &: M \to T \\
\tau \circ \mathsf{sr} &: M \to T
\end{aligned}$$

Here τ^a denotes the curried typing function given by $\tau^a(b) = \tau(a, b)$. For each function, we obtain a security property corresponding to each of the basic forms of opaqueness:

- k-value opaqueness;
- absolute value opaqueness;
- Z-value opaqueness (for subsets Z arising canonically in the function range);
- image opaqueness;
- kernel opaqueness.

Sender function $\mathsf{s} : M \to A$

1. *k-value opaqueness of sender function* $\mathsf{s} : M \to A$. Attacker can only discern the originator of any given conversation up to a lineup of size k. This property is sometimes referred to as (sender) ***k-anonymity*** or ***lineup anonymity*** [SS98, Mal01].

2. *absolute value opaqueness of sender function* $\mathsf{s} : M \to A$. Attacker cannot infer the identities of the senders since every agent is a plausible sender for every observed conversation. This property corresponds to ***absolute sender anonymity***, and is implemented, *e.g.*, by persistent pseudonyms such as those provided by Web-based email. While messages sent from a Web-based email account can be traced to that account, the real identity of the person behind the account remains absolutely anonymous since every person in the world with Internet access is in the lineup of potential senders. Chaum's DC-nets [Cha85, Cha88] can also provide absolute sender anonymity but only if every potential sender in the world uses the DC-net for communication; otherwise, the attacker can eliminate agents not participating in the protocol from the lineup, and the strongest property provided is *type-anonymity* (see below).

   The attacker is still permitted to discern other information such as whether two conversations originated from the same (unknown) sender. This has implications for forward security. If persistent pseudonyms have been used to ensure absolute sender anonymity, and the mapping from the pseudonym to the real identity is compromised even for a single conversation, the attacker can automatically infer the identity of the sender for all other conversations originating from the same pseudonym. In section 4.8, we describe an extension to the protocol graph model of section 4.1 that allows direct specification of unlinkability of pseudonyms and real identities.

3. $(\tau^a)^{-1}(t)$-*value opaqueness of sender function* $\mathsf{s} : M \to A$. The attacker may learn the type of the sender with respect to the recipient, but not the sender's identity. We will call this flavor of anonymity ***type-anonymity***. For example, if the communication tower of a mobile phone network executes observably different protocols with subscribers and roaming users, the attacker may learn that a particular phone is a subscriber of the network without learning the phone's identity.

   Type-anonymity effectively reduces the lineup of plausible senders to an entire type. Therefore, the system is vulnerable if the type is small. For example, if there are only two people in the world using a particular secure email protocol, then observing an instance of the protocol is sufficient to reduce the sender lineup to 2.

   All anonymity systems that involve routing messages through specialized servers such as those based on Chaum's MIX-nets [Cha81, Ano], onion routing [SGR97], Crowds [RR98], *etc.* provide type-anonymity, since they require users to engage in an observably distinct communication protocol. This is also true for approaches based on DC-nets [Cha85, Cha88].

4. $(\mathsf{im}\,\mathsf{s})$-*value opaqueness of sender function* $\mathsf{s} : M \to A$. Session-level sender anonymity. The attacker may know the entire set of senders, but is unable to link conversations to the identities of the senders. This flavor of anonymity is provided, *e.g.*, by online chatrooms. An observer may be able to see the list of all users logged on at the time a particular message appeared, but cannot find out who actually posted the message.

   Since in this case the attacker is able to observe whether a particular agent is engaged in communication or not, this form of anonymity reduces the lineup to the set of session participants and is thus vulnerable if the set is small. For example, if only one person was logged on at the time a message appeared, the attacker can infer the identity of the sender.

5. *image opaqueness of sender function* $\mathsf{s} : M \to A$. This flavor of anonymity is required (but is not sufficient) for ***unobservability*** [PW87, PPW91, PKS01]. For image opaqueness to hold, there should be no agent whom the attacker can pinpoint as one of the senders. This implies that for every configuration in which some agent sent something, even if the contents and the destination of the message are protected, there should be another, observationally indistinguishable configuration in which the same agent did *not* send anything. In other words, the act of sending a message should be unobservable: an observer should not be able to

determine whether a particular agent is engaged in an anonymizing protocol or not.

Note that even MIX-nets with dummy traffic are insufficient to guarantee this property, unless users access MIX-nets via secure "nymservers" [GWB97] that create online identities for them which are unlinkable to their real identities.

6. *kernel opaqueness of sender function* $\mathsf{s} : M \to A$. This type of anonymity is required (but not sufficient) for ***untraceability*** [SMA95]. An observer should not be able to determine whether two conversations have the same sender. This prevents the attacker from linking multiple instances of communication as originating from the same source and/or tracing an agent through multiple conversations. Forward security requires kernel opaqueness so that if one conversation is compromised, others are not affected.

   To implement this form of anonymity, it is necessary to create a new identity each time a user engages in a conversation on an observable network link.

Recipient function $\mathsf{r} : M \to A$

Recipient anonymity properties are symmetric to sender properties enumerated above.

Sender-recipient function $\mathsf{sr} : M \to A \times A$

1. *k-value opaqueness of sender-recipient function* $\mathsf{sr} : M \to A \times A$. For $k = 2$, this form of anonymity means that the attacker may be able to determine the sender or the recipient, but not both at the same time. For $k > 2$, the lineup of plausible sender-recipient pairs should contain at least k candidates. No system that we are aware of provides this form of anonymity alone. There are systems, *e.g.*, Mixmaster-type remailers [Cot96], that provide stronger anonymity guarantees, which imply k-value opaqueness for sr, among other properties.

2. *absolute value opaqueness of sender-recipient function* $\mathsf{sr} : M \to A \times A$. Equivalent to absolute sender and absolute recipient anonymity.

3. *$\tau^{-1}(t)$-value opaqueness of sender-recipient function* $\mathsf{sr} : M \to A \times A$. It is possible to observe the type of communication (*i.e.*, relationship between communicating agents), but not their identities. For example, the attacker may be able to determine that a particular email is sent by some clinic to some patient without being able to tell who the clinic or the patient is. This property is dual to relationship anonymity, in which the attacker is permitted to learn the identities of communicating agents but not their relationship.

4. $(\mathsf{im}\,\mathsf{sr})$-*value opaqueness of sender-recipient function* $\mathsf{sr} : M \to A \times A$. Session-level relationship anonymity. This is provided by MIX-nets without dummy traffic. The attacker can observe whether communication occurred, and who participated in it, but cannot link senders with recipients. For each message on an

incoming link, an observer can determine the sender, and for each outgoing message, the recipient, but since messages are re-encrypted by the MIX, an observer cannot determine *both* the sender and the recipient for any message.

5. *image opaqueness of sender-recipient function* $\mathsf{sr} : M \to A \times A$. Universal relationship anonymity. Any observation of the system by the attacker should be consistent with any given pair of agents communicating or not communicating. This form of anonymity can be implemented by MIX-nets with constant dummy traffic and pseudonyms for all participants.

6. *kernel opaqueness of sender-recipient function* $\mathsf{sr} : M \to A \times A$. It should not be possible to observe whether two conversations followed the same path (*i.e.*, went from the same sender to the same recipient) even if the identities of the sender and recipient are unknown. Implementation requires randomly generated identities.

Curried typing function $\tau^a : A \to T$

Fixing a particular agent a, opaqueness properties of the curried typing function $\tau^a : A \to T$ model hiding information about relationships between a and other agents.

1. *k-value opaqueness of curried typing function* $\tau^a : A \to T$. The lineup of possible relationships has at least k candidates. For example, let $T = \{\mathsf{Danish}, \mathsf{Dutch}, \mathsf{Spanish}, \mathsf{Italian}\}$, with $\tau(a, b) = l$ denoting the fact that agent a talks to agent b in language $l \in T$. Use of a primitive cipher may result in the attacker being able to discern whether a talks to b in a Germanic language (Danish or Dutch), or in a Romance language (Spanish or Italian). Thus τ^a is 2-value opaque.

2. *absolute value opaqueness of curried typing function* $\tau^a : A \to T$. A form of privacy: observations of the system are consistent with any possible relationship between a and any other agent.

3. $(\mathsf{im}\,\tau^a)$*-value opaqueness of curried typing function* $\tau^a : A \to T$. The lineup of possible relationships can be reduced to those that manifested themselves in actual communication. Consider an online chatroom where messages are posted in different languages. An observer may be able to tell that there are posters who communicate in French and German, but not who they are. An observer can also determine that there are no posters communicating in Russian.

4. *image opaqueness of curried typing function* $\tau^a : A \to T$. For any type of relationship, it should not be possible to determine whether there exist any agents that have this relationship with a. For example, if a is a clinic treating various illnesses, it should not be possible to infer whether it has patients suffering from a particular illness.

5. *kernel opaqueness of curried typing function* $\tau^a : A \to T$. Inability to determine whether two agents have the same relationship with a. Required for forward security.

Typing function $\tau : A \times A \to T$

These properties are similar to those for τ^a. The main difference is that they are not defined with respect to a particular agent, but range over all relationships existing in the world.

1. *k-value opaqueness of typing function* $\tau : A \times A \to T$. Any observed relationship between agents can plausibly belong to any of k types.

2. *absolute value opaqueness of typing function* $\tau : A \times A \to T$. A form of privacy: an observer cannot determine the relationship between any two agents.

3. $(\mathsf{im}\,\tau)$*-value opaqueness of typing function* $\tau : A \times A \to T$. An observer can tell which types are empty (*e.g.*, there are no smallpox patients in the world) and which are not (*e.g.*, there exist malaria patients), but no more. For example, even if he observes an agent communicating with a clinic, he cannot tell whether the agent is a patient or not, or what he is being treated for.

4. *image opaqueness of typing function* $\tau : A \times A \to T$. It is impossible to determine which relationships exist (*i.e.*, the type is non-empty) and which do not.

5. *kernel opaqueness of typing function* $\tau : A \times A \to T$. Inability to determine that two pairs of agents have the same relationship. For instance, this prevents the attacker from learning that two pairs of (unknown) agents are communicating in the same (unknown) cipher. If the cipher is broken for one pair, it will not be automatically broken for the other.

Composite functions $\tau \circ \mathsf{sr},\ \tau^a \circ \mathsf{s},\ \tau^a \circ \mathsf{r} : M \to T$

Atomic opaqueness properties of the three composites $\tau \circ \mathsf{sr}$, $\tau^a \circ \mathsf{s}$, and $\tau^a \circ \mathsf{r}$ can be defined in a similar manner. We mention explicitly only value opaqueness of the composite $\chi = \tau \circ \mathsf{sr} : M \to T$, as it is important for our analysis of the distinction between anonymity and privacy (see section 4.7). Each form of value opaqueness of χ asserts the observer's inability to discern the type of the conversations for which he knows the sender and the recipient, *or* inability to determine the sender and the recipient of the conversations for which he knows the type. For example, privacy (defined as relationship anonymity) is guaranteed even if the attacker intercepts a message from a known agent to a known clinic as long as he cannot determine whether the message is from a patient or a non-patient.

One of the more subtle distinctions in the above taxonomy is that between $(\mathsf{im}\,\mathsf{s})$-value opaqueness and image opaqueness of sender function s. This distinction models the difference between session-level and universal anonymity. (The latter is sometimes called *unobservability* [PKS01]). With session-level anonymity, an observer can determine the identities of the agents who are actually engaged in conversations, but not link these identities to specific conversations. This is an "Agatha Christie" form of anonymity:

there is a finite set of suspects who were in the house at the time of the murder. Universal anonymity (unobservability) is a stronger property. An observer cannot determine whether or not any given agent was engaged in communication. This is a "Georges Simenon" form of anonymity: a body is found, and any of the several million city residents may or may not have been involved.

4.4 Hiding identities vs. hiding relationships

In section 4.3, we gave a taxonomy of atomic anonymity and privacy properties. Each one of them deals with what is sometimes called ***linkability*** [PKS01] of various system elements such as conversations and their senders and recipients. Opaqueness of the corresponding function models the attacker's inability to *link* the items, that is, to obtain knowledge about their relationship. In section 4.5, we show how properties provided by many proposed anonymity systems can be characterized as one of the atomic properties from section 4.3 or a combination thereof. Following the methodology of section 3.6, each combination of opaqueness-based properties can be converted into a compound logical predicate over the observational equivalence classes of the underlying calculus, which may then be amenable to verification.

Properties defined in section 4.3 fall naturally into two general classes: those dealing with the unlinkability of conversations and agents (*e.g.*, "did this phone make this call?"), and those dealing with the unlinkability of agents (*e.g.*, "is this phone a subscriber on this network?"). Roughly, the former can be interpreted as *anonymity* properties since they are concerned with hiding identities of agents who performed certain actions. The latter can be interpreted as *privacy* properties since they are concerned with hiding relationships between agents. Our formalization provides a crisp mathematical boundary between the two classes of properties. In section 4.7 below, we discuss this distinction in more detail, and argue that, contrary to popular belief, anonymity is neither necessary, nor sufficient for privacy. It is possible for a communication protocol to guarantee privacy of relationships without hiding identities, and vice versa: some anonymous protocols may be exploited to learn protected information about relationships between agents.

While the protocol graphs defined in section 4.1 are sufficient to reason about many flavors of anonymity and privacy, some forms of anonymity require a richer data structure in order to model them correctly. In section 4.8, we demonstrate how pseudonymity can be modeled by extending protocol graphs with mappings from pseudonyms to real identities. The function view framework and opaqueness-based definitions of information hiding properties are still applicable, but the functions to which they are applied must be chosen differently.

4.5 Existing systems

Anonymity is an active area of computer security research, and dozens of protocols and systems have been proposed to implement various notions of anonymity. Unfortunately, there is no universally accepted terminology in the field, and different authors

Property	Informal definition	Opaqueness definition
Sender untraceability	Attacker may not know anything about senders (*e.g.*, MIX-net [Cha81, Dai95] with nymservers and dummy traffic, [SMA95])	s is absolutely value opaque and kernel opaque
Absolute sender anonymity	Attacker may not know who the senders are but may know whether 2 conversations originate from the same (unknown) agent (*e.g.*, Web-based email)	s is absolutely value opaque
Sender k-anonymity	Attacker may only know senders up to a lineup of size k (*e.g.*, [SS98, Mal01])	s is k-value opaque
Recipient properties are obtained from the above by substituting r for s		
Blender anonymity	Attacker may know which agents sent and received communications, but any sender (recipient) is a plausible source (destination) for any conversation (*e.g.*, MIX-nets [Cha81, SGR97] without dummy traffic)	s is $\mathsf{im}\,\mathsf{s}$-value opaque; r is $\mathsf{im}\,\mathsf{r}$-value opaque
Conversation-agent 2-unlinkability	Attacker may not know both sender and recipient for any conversation	sr is 2-value opaque
Privacy (relationship anonymity)	Attacker may not determine the relationship between any 2 agents (*e.g.*, [SMA95, Aba02])	τ is 2-value opaque

Table 1: Definitions of anonymity and privacy based on opaqueness.

invest terms such as *unlinkability*, *unobservability*, *privacy*, *etc.* with different meaning. Our descriptions are not intended to be a definitive dictionary of anonymity terms (cf. [PKS01]), but rather an intuitive explanation of how they are interpreted for the purposes of our case study. Since our framework is non-probabilistic, we do not attempt to model probabilistic notions of anonymity and/or quantify the level of identity protection provided by systems such as Crowds [RR98] and Freenet [CSWH00]. Nonetheless, as remarked in section 2.5, our technique is still applicable if the chosen process specification formalism provides a suitable observational equivalence relation that abstracts away from probabilities.

We give brief informal descriptions of several common anonymity properties with references to relevant protocols in Table 1. We also show how they correspond to atomic properties (or combinations thereof) from the taxonomy of section 4.3, by defining each property formally in terms of opaqueness of the underlying system attributes. Since the equivalence relation $\stackrel{\mathcal{P}}{\sim}$ on protocol graphs partitions $\mathbb{C}_{M,A,T}$ into equivalence classes, we can apply Proposition 3 and derive the predicates that the classes must satisfy in order for the protocol to satisfy the corresponding property. The predicates are listed in Table 2. Figure 2 shows implications between the various anonymity properties. A solid arrow denotes implication, a dashed arrow conjunction.

A common design is based on Chaum's MIX-nets [Cha81] and involves a network of anonymizing servers [Dai95, Cot96, SGR97]. This approach provides identity protection in a situation where the "Big Brother" attacker may observe traffic on the wires connecting everyone's computer to the network, but all communications are encrypted and travel through a secure server(s) which blends them together and distributes to their respective destinations in a manner which prevents the attacker from linking the origin and destination of any given message. Without constant dummy traffic to make all conversations unobservable, MIX-nets only provide blender anonymity: the attacker is able to observe all senders and all recipients, but cannot determine both the sender and the recipient for any message.

Note that even if the mapping from conversations to sender-recipient pairs is secure, *i.e.*, sr is opaque, the attacker may still be able to infer who is conversing with whom. Consider, for example, a protocol employing a central anonymizing blender and the following protocol graph:

$$\begin{tikzcd} a \arrow[r, "m"] & c \\ b \arrow[ur, shift left, "p"] \arrow[ur, shift right, "q"'] \arrow[r, "n"'] & d \end{tikzcd}$$

If the attacker is permitted to observe the senders and the recipients, he can infer that b is conversing with c by observing that b is involved in 2 conversations, while d is involved in only 1 conversation. Therefore, one of b's conversations must be with c. Even though the attacker's ability to obtain this knowledge could be considered an attack on anonymity, it is *not* a violation of blender anonymity. For any given conversation, the attacker cannot determine the sender or the recipient. He knows that *either* p, *or* q is

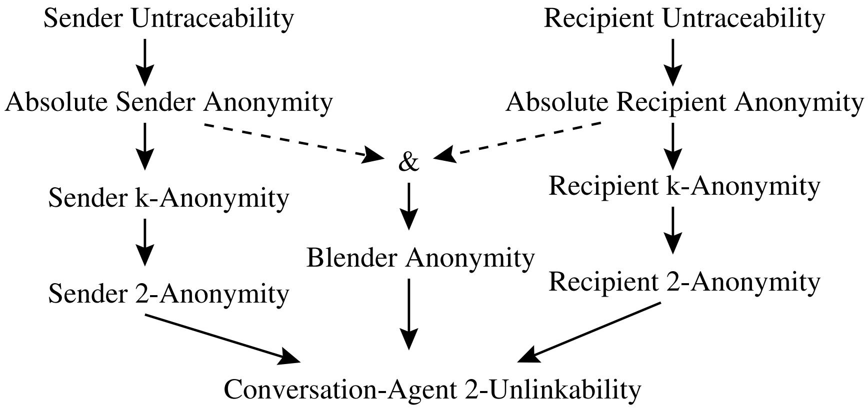

Figure 2: Hierarchy of anonymity properties

an instance of b conversing with c, but if communication is routed through a blender, he cannot know which one.

This illustrates the importance of posing the correct anonymity requirement. To hide the fact that two agents are communicating, the needed property is relationship anonymity, where relationship can be modeled by $\tau(x, y) = \texttt{true}$ *iff* x and y are communicating. This is a stronger property than simple blender anonymity. In particular, it requires that an agent's participation in communication be unobservable, in order to prevent the attacker from counting conversations with the MIX on each link. Typically, this is implemented by adding dummy traffic to the system. Even in this case, unless the MIX is accessed through a nymserver, the strongest property that can be guaranteed is type-anonymity, where the type is the set of all people using the anonymous communication software, since the attacker can observe which agents are running the software (*e.g.*, by observing traffic to the MIX from their computers, without knowing whether the traffic is dummy or real).

4.6 Privacy as relationship anonymity

Many definitions of privacy, often contradictory, can be found in the literature [CB95, SMA95, STRL00]. The colloquial meaning of privacy concerns either the secrecy of personal data (which is beyond the scope of this research), or the secrecy of relationships between agents. In the context of secure communication systems, the latter is of paramount importance. For example, it should not be possible for an observer to infer that a particular person is patient of a certain medical clinic, that the user of a particular mobile phone has an account with a certain bank, *etc.*

In the protocol graph framework described in section 4.1, relations between agents are formalized as a type assignment $\tau : A \times A \to T$. While anonymity properties have to do with hiding information about sr (*i.e.*, protecting identities of the endpoints of conversations), privacy has to do with hiding information about τ. In Table 1, privacy is formally defined as the requirement that τ is 2-value opaque. The corresponding

Absolute sender anonymity	$\forall C \in \mathbb{C}_{M,A,T}.\forall m \in M.\forall a \in A.\exists C' \in \mathbb{C}_{M,A,T}.$ $C' \overset{\mathcal{P}}{\sim} C \wedge \mathsf{s}_{C'}(m) = a$
Sender untraceability	*Absolute sender anonymity* $\wedge$ $\forall C \in \mathbb{C}_{M,A,T}.\forall m \in M.\forall n \in M.\mathsf{s}_C(n) = \mathsf{s}_C(m).$ $\exists C' \in \mathbb{C}_{M,A,T}.C' \overset{\mathcal{P}}{\sim} C \wedge \mathsf{s}_{C'}(m) \neq \mathsf{s}_{C'}(n)$
Sender k-anonymity	$\forall C_1 \in \mathbb{C}_{M,A,T}.\forall m \in M.\exists C_2,\ldots,C_k \in \mathbb{C}_{M,A,T}.$ $C_i \overset{\mathcal{P}}{\sim} C_j \wedge [i \neq j \Rightarrow \mathsf{s}_{C_i}(m) \neq \mathsf{s}_{C_j}(m)]$
Recipient properties are obtained from the above by substituting r *for* s	
Blender anonymity	$\forall C \in \mathbb{C}_{M,A,T}.\forall m \in M.$ $[\forall a \in \mathsf{im}\,\mathsf{s}_C.\exists C' \in \mathbb{C}_{M,A,T}.C' \overset{\mathcal{P}}{\sim} C \wedge \mathsf{s}_{C'}(m) = a] \wedge$ $[\forall a \in \mathsf{im}\,\mathsf{r}_C.\exists C' \in \mathbb{C}_{M,A,T}.C' \overset{\mathcal{P}}{\sim} C \wedge \mathsf{r}_{C'}(m) = a]$
Conversation-agent 2-unlinkability	$\forall C \in \mathbb{C}_{M,A,T}.\forall m \in M.\exists C' \in \mathbb{C}_{M,A,T}.$ $C' \overset{\mathcal{P}}{\sim} C \wedge \mathsf{sr}_{C'}(m) \neq \mathsf{sr}_C(m)$
Privacy	$\forall C \in \mathbb{C}_{M,A,T}.\forall m \in M.\exists C' \in \mathbb{C}_{M,A,T}.$ $C' \overset{\mathcal{P}}{\sim} C \wedge \tau_{C'}(m) \neq \tau_C(m)$

Table 2: Predicates on equivalence classes for anonymity and privacy.

predicate on observational equivalence is listed in Table 2.

In some situations, $\tau(a, b)$ may influence which protocol is executed between a and b when they communicate on an observable network. Opaqueness of τ does *not* require that the protocols be observationally equivalent for all values of $\tau(a, b)$. Consider the following protocol between an agent A and a medical clinic website C ($\mathsf{pubk}(X)$ is agent X's public key, N_X is a fresh random number generated by X):

$$\begin{array}{ll}
A \to C: & \{\text{“}hi\text{”}, A, N_A\}_{\mathsf{pubk}(C)} \\
C \to A \text{ (patient)}: & \{\text{“}ok\text{”}, C, N_C\}_{\mathsf{pubk}(A)} \\
C \to A \text{ (nonpatient)}: & N'_C \\
A \to C \text{ (patient)}: & \{N_C\}_{\mathsf{pubk}(C)}
\end{array}$$

In this protocol, the attacker can tell the difference between the clinic-patient and clinic-nonpatient protocols (the former has 3 messages, the latter 2). In the language of section 4.3, the property provided by the protocol is $(\mathsf{im}\,\tau^{clinic})$-value opaqueness.

The attacker cannot, however, determine the identity of the person communicating with the clinic, even though he knows whether this unknown person is a patient or not. There is, therefore, no agent a—other than the attacker himself and his allies—for which the attacker can determine the value of $\tau(a, clinic)$. The protocol preserves relationship anonymity, and thus privacy.

4.7 Independence of anonymity and privacy

Contrary to the popular idea that anonymity is necessary for privacy [GWB97, Hug93], our formalization demonstrates that privacy of communications requires *either* hiding $\mathsf{sr} : M \to A \times A$ (*i.e.*, anonymity), *or* hiding $\chi = \tau \circ \mathsf{sr}$. If neither is hidden, then privacy fails (the converse is not true: even when both sr and χ are hidden, relationship anonymity may still fail). For example, if the attacker can establish that $\mathsf{sr}(m) = \langle a, \mathit{clinic}\rangle$ for some conversation m and agent a (*i.e.*, the protocol is not anonymous in any sense) and $\chi(m) = \mathsf{patient}$ (*i.e.*, the protocols executed by the clinic with patients and nonpatients are observably different), then the attacker can infer that $\tau(a, \mathit{clinic}) = \mathsf{patient}$, thus violating a's privacy.

For some simple relationships anonymity may be sufficient to guarantee privacy. For example, if $\tau(a, b) = \mathsf{true}$ when a and b are conversing, *i.e.*, $\exists m \in M$ s.t. $\mathsf{sr}(m) = \langle a, b\rangle$, and false otherwise, then sender anonymity will ensure relationship anonymity for a. In general, however, anonymity does not guarantee privacy even if χ is opaque, nor does privacy require anonymity. Consider two artificial protocols between agent A and clinic C (K_A is a fresh key generated by A):

	Protocol 1	**Protocol 2**
$A \to C$:	$\{\text{“}hi\text{”}, A\}_{\mathsf{pubk}(C)}$	$\{\text{“}hi\text{”}, A, K_A\}_{\mathsf{pubk}(C)}$
$C \to A$ (patient) :	$\{\text{“}ok\text{”}, N'_C\}_{\mathsf{pubk}(A)}$	$\{\text{“}ok\text{”}\}_{K_A}$
$C \to A$ (nonpatient) :	N_C	N_C

If the public-key encryption function happens to be deterministic, Protocol 1 provides relationship anonymity without sender or recipient anonymity. The attacker can precompute a table with all possible values of the $A \to C$ message using all agent names he knows. Whenever he observes a message on the network, he can infer who the sender is by looking up the message in the table. Still, there is no way for the attacker to determine whether A (or any other agent except the attacker and his allies) is a patient of C. Note that $\chi = \tau \circ \mathsf{sr}$ is opaque in Protocol 1.

Protocol 2 provides absolute sender anonymity (untraceability, in fact) and keeps χ opaque, but does not guarantee privacy. The attacker cannot determine who A is by observing the protocol, nor can he figure out whether the unknown sender is a patient or not. At the same time, due to the lack of authentication, the attacker can exploit the protocol to find out whether any given agent A' is a patient by creating a message $\{\text{“}hi\text{”}, A', K_e\}_{\mathsf{pubk}(C)}$, where K_e is known to the attacker, and sending it to C pretending to be A'. Since the attacker can tell the difference between $\{\text{“}ok\text{”}\}_{K_e}$ and N_C, he can infer from C's response whether A' is a patient. The attack on Protocol 2 is similar to the attack described by Abadi in the context of private authentication [Aba02].

4.8 Pseudonymity

We have already remarked in section 2.5 how the notion of function view can be extended to express a larger class of information hiding properties. For example, one could add a fourth component to the function knowledge triple to record information about the cardinality of inverse images of points in the range. Correspondingly, one would obtain an additional form of opaqueness.

Another form of extension arises at the level of the data structure over which we express our forms of opaqueness. In the case study we used a specific protocol graph with three functions (sender, recipient, and type) to systematically generate a space of anonymity and privacy properties. By design, this space included many of the properties found in the literature. The underlying graph, however, limits the set of properties that can be arrived at in this manner.

Other security situations may demand a different protocol graph. For example, consider *pseudonymity* [ISO99, PKS01]. In a pseudonymity system, agents employ intermediate identities or *pseudonyms* from which they send and receive communications. Each agent may own multiple pseudonyms, *e.g.*, numerous free Web-based email addresses. This richer setting admits a richer class of anonymity and privacy properties. Following the same general methodology as in section 4.3, we can explore the class systematically using opaqueness of the functions in the data structure associated naturally with this setting:

$$M \overset{\mathsf{sr}_C}{\to} P \times P \overset{\tau_C}{\to} T$$

$$P \overset{\omega_C}{\to} A$$

where M, A and T are as before (conversations, agents, and types), P is a set of pseudonyms, sr_C assigns sender-recipient pairs of pseudonyms to conversations, τ_C is a typing, and ω_C determines ownership of pseudonyms. Now in order to systematically generate a space of anonymity and privacy properties, one has a richer basis of functions than in our original case study. In addition to the familiar s, r, sr and τ, one has $\omega : P \to A$, $\omega \circ \mathsf{s} : M \to A$, $\omega : M \to A$, and $(\omega \times \omega) \circ \mathsf{sr} : M \to A \times A$. A systematic exploration of the various forms of opaqueness of these functions, in the manner of section 4.3, yields an array of formal anonymity and privacy properties for pseudonymity systems.

The pseudonymity example complements our original case study by emphasizing our general methodology. For any security scenario, an appropriate choice of the underlying abstract data structure (above, protocol graphs) automatically yields a range of atomic information hiding properties as opaquenesses of the functions (attributes) comprising the data structure.

5 Conclusion

We developed a modular framework for formalizing properties of computer systems in which an observer has only partial information about system behavior. Figure 1 summarizes the difference between our approach and conventional approaches based on process algebra or epistemic logic. Our techniques combine the benefits of the knowledge-based approach, namely, natural specification of information hiding properties, with those of the process algebra-based approach, namely, natural specification of system behavior. Furthermore, our framework is parametric, leaving open the choice of underlying process algebra and logic.

We proposed the notion of a *function view*, a succinct mathematical abstraction of partial knowledge of a function. Remarkably, the three attributes of a function view —

graph, kernel, and image — suffice to model many forms of partial knowledge an observer may have about a function of interest. We demonstrated how any formalism for system specification that provides an equivalence relation on system configurations induces function views, and how information hiding properties could be stated naturally in terms of *opaqueness* of these views.

In the case study of section 4 we employed our framework to systematically circumscribe, formalize and classify a range of anonymity and privacy properties. One important result of this systematic exploration is a crisp mathematical distinction between anonymity and privacy of communications (the latter interpreted as relationship anonymity), which were shown to be independent (section 4.7) in a rigorous technical sense.

The independence of anonymity and privacy has significant implications for public policy in the area of personal information protection on the Internet. It shows that care must be exercised when deciding on the appropriate mechanism for privacy. In particular, we conclude that anonymity is neither necessary, nor sufficient to ensure that personal information about one's relationships with other people and organizations is protected in online communications.

Acknowledgements We are grateful to Martín Abadi for very helpful discussions and to Veronique Cortier for pointing out that distinguishability of protocols corresponding to different relationships between agents does not necessarily imply a failure of privacy or anonymity. Thanks to Julien Basch for help finding the counterexample to associativity of function view composition.

References

[Aba02] M. Abadi. Private authentication. In *Proc. Workshop on Privacy-Enhancing Technologies (PET)*, 2002. To appear.

[AG99] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi-calculus. *Information and Computation*, 148(1):1–70, 1999.

[AJ94] S. Abramsky and A. Jung. Domain theory. In *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Oxford University Press, 1994.

[Ano] `http://www.anonymizer.com`.

[AR02] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 15(2):103–127, 2002.

[BNP99] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Proc. 14th IEEE Symposium on Logic in Computer Science (LICS)*, pages 157–166, 1999.

[CB95] D. Cooper and K. Birman. Preserving privacy in a network of mobile computers. In *Proc. IEEE Symposium on Security and Privacy*, pages 26–38, 1995.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[Cha85] D. Chaum. Security without identification: transaction systems to make Big Brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[Cha88] D. Chaum. The Dining Cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.

[Cot96] L. Cottrell. Frequently asked questions about Mixmaster remailers, 1996. `http://www.obscura.com/~loki/remailer/mixmaster-faq.html`.

[CSWH00] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 46–66. Springer-Verlag, 2000.

[Dai95] Wei Dai. Pipenet. Post to the `cypherpunks` mailing list, February 1995.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[FHMV95] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[GM82] J. Goguen and J. Meseguer. Security policy and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[Gun92] C. Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, 1992.

[GWB97] I. Goldberg, D. Wagner, and E. Brewer. Privacy-enhancing technologies for the Internet. In *Proc. 42nd IEEE Spring COMPCON*, pages 103–109, 1997.

[Hin62] J. Hintikka. *Knowledge and Belief.* Cornell University Press, 1962.

[HT93] J. Halpern and M. Tuttle. Knowledge, probability, and adversaries. *J. ACM*, 40(4):917–962, 1993.

[Hug93] E. Hughes. A Cypherpunk's Manifesto. `http://www.activism.net/cypherpunk/manifesto.html`, 1993.

[ISO99] ISO. IS 15408. `http://www.commoncriteria.org/`, 1999.

[Kri63] S. Kripke. A semantic analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Matematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[LMMS99] P. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *Proc. World Congress on Formal Methods*, volume 1708 of *LNCS*, pages 776–793. Springer-Verlag, 1999.

[Mal01] D. Malkhi. Private communication, 2001.

[PKS01] A. Pfitzmann, M. Köhntopp, and A. Shostack. Anonymity, unobservability, and pseudonymity - a proposal for terminology. Manuscript (available from A. Pfitzmann), June 2001.

[PPW91] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-MIXes–untraceable communication with very small bandwidth overhead. In *Proc. IFIP/Sec '91*, pages 245–258, 1991.

[PW87] A. Pfitzmann and M. Waidner. Networks without user observability. *Computers and Security*, 6(2):158–166, 1987.

[RR98] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[Sch96] S. Schneider. Security properties and CSP. In *Proc. IEEE Symposium on Security and Privacy*, pages 174–187, 1996.

[Sco72] D.S. Scott. Continuous lattices. In E. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, Lecture Note in Mathematics 274, pages 97–136. Springer-Verlag, 1972.

[SGR97] P. Syverson, D. Goldschlag, and M. Reed. Anonymous connections and onion routing. In *Proc. IEEE Symposium on Security and Privacy*, pages 44–54, 1997.

[SMA95] D. Samfat, R. Molva, and N. Asokan. Anonymity and untraceability in mobile networks. In *Proc. 1st Annual International Conference on Mobile Computing and Networking (MOBICOM)*, pages 26–36, 1995.

[SP01] E. Sumii and B.C. Pierce. Logical relations for encryption. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 256–269, 2001.

[SS96] S. Schneider and A. Sidiroupoulos. CSP and anonymity. In *Proc. 4th European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *LNCS*, pages 198–218. Springer-Verlag, 1996.

[SS98] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL-98-03, Computer Science Laboratory, SRI, 1998.

[SS99] P. Syverson and S. Stubblebine. Group principals and the formalization of anonymity. In *Proc. World Congress on Formal Methods*, volume 1708 of *LNCS*, pages 814–833. Springer-Verlag, 1999.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[STRL00] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an analysis of onion routing security. In *Proc. International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 96–114. Springer-Verlag, 2000.