A Dynamic Obfuscation Framework for Security and Utility

Andrew Wintenberg
awintenb@umich.edu
The University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

Matthew Blischke*
blischke@ucsb.edu
The University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

Stéphane Lafortune
stephane@umich.edu
The University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

Necmiye Ozay
necmiye@umich.edu
The University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

ABSTRACT

Obfuscation can be used by dynamic systems to ensure private and secure communication over networks vulnerable to eavesdroppers. Balancing the utility of sending information to intended recipients and privacy by hiding information from unintended recipients presents an interesting challenge. We propose a new framework for obfuscation that includes an inference interface to allow intended recipients to interpret obfuscated information. We model the security of the obfuscation with opacity, a formal notion of plausible deniability. Using techniques from distributed reactive synthesis, we show how to automatically design a privacy-enforcing obfuscator along with the inference interface that is given to intended recipients to use as a "key". We demonstrate this approach by enforcing privacy while maintaining utility in a contact tracing model.

KEYWORDS

Privacy and Security, Obfuscation, Opacity, Discrete Event Systems

1 INTRODUCTION

Recently, the number of networked cyber-physical systems has been increasing dramatically in the form of autonomous vehicles, the smart grid, location-based services, medical monitoring, and other applications. As these systems transmit potentially sensitive information to recipients, there are often strict requirements for privacy and security. However, many commonly used networks are vulnerable to eavesdropping by unintended recipients. For example, wireless networks are inherently accessible to the general public. To address this issue, techniques from formal methods have been used to understand information flow within these systems, for example observational determinism [11] or non-interference [2]. In particular the information flow property of *opacity* has been widely used to express notions of privacy and security of cyber-physical systems [27, 28]. In words, opacity captures the notion of *plausible deniability*: opacity holds if unintended recipients cannot deduce sensitive information. A general overview of opacity is provided in [12].

In this setting, a common problem is given an existing system called the *plant* that is not opaque, how do we *enforce* opacity upon it? A variety of mechanisms have been proposed for opacity enforcement. For example, supervisory control enforces opacity by restricting behavior that would reveal sensitive information when transmitted to the network [6, 23]. However, it may not be practical to alter or restrict an existing plant's behavior, e.g., human behavior in a cyber-physical system. Alternatively, we consider obfuscation which enforces opacity by altering the information transmitted to the network. In addition to privacy, obfuscation must also maintain the utility of the system in granting access to information for some intended recipients. In [25], the selective insertion and deletion of outputs with *edit functions* can effectively hide secrets from anyone on the network. However, the utility of this method is limited as these methods do not distinguish between intended and unintended recipients. Hence information that is available to some is available to all. Likewise, many techniques for achieving differential privacy [7] are similar to obfuscation in adding noise to the outputs of the system to hide information, but again do not distinguish recipients.

In this paper we propose an obfuscation framework that allows an intended recipient to infer sensitive information that cannot be deduced by unintended recipients. Similar to encryption, this is possible by designing a "key" that is provided to the intended recipient to recover information about the plant from their obfuscated observations. Whereas encryption achieves privacy by conspicuously altering data to ensure it is impossible or at least computationally difficult to recover, our proposed method of obfuscation achieves privacy by inconspicuously altering data to *deceive* recipients with partial knowledge of the system. In this sense, our goals for obfuscation are orthogonal to those of encryption. We provide an automatic method for the simultaneous design of obfuscation and inference policies for dynamic systems subject to security and utility requirements. In this work, we focus on the networked aspect of cyber-physical systems. In this setting, plants can be modeled by finite automata over which we can specify requirements with temporal logics. We consider obfuscation policies similar to edit functions which can produce a positive number of outputs given a single input from the plant. By modeling the obfuscation and inference policies as processes in a distributed system, we can leverage techniques from distributed synthesis [10] to design solutions with formal guarantees of privacy and utility. We motivate our solution to this problem with the following simple example.

Example 1.1. *Consider a company operating a research facility whose layout is depicted in Figure 1. Smart devices report employee's locations throughout the building to a server over an internal network. However, the devices' accuracies are limited and only report an approximate location represented by a region of the building: $\mathcal{R}$ is the lobby, $\mathcal{T}$ is the offices, and $\mathcal{U}$ is the electronics lab and chemical lab. The company is concerned this information may reveal to their*

*Currently at ECE Department at the University of California, Santa Barbara

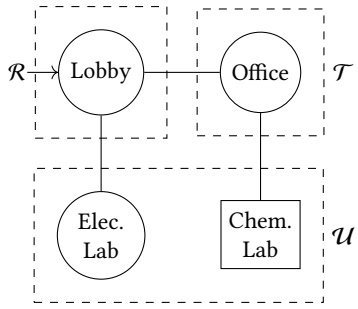

Figure 1: The graph $\Gamma = (\mathcal{V}, \mathcal{E})$ that represents the locations $\mathcal{V}$ and the physical paths between them $\mathcal{E}$. Regions $\mathcal{R}, \mathcal{T}, \mathcal{U}$ represent the approximate locations the smart device can report and partition the space $\mathcal{V}$. The chemistry lab is considered to be a *secret* location.

competitors how they allocate their resources, i.e., when an employee enters the chemistry lab. However, the company does not want to restrict employees' movements or alter their schedules. While encryption can secure this information over the network, it alerts competitors to the existence of a secret and may prompt them to investigate using other means. Instead the company chooses to deceive competitors by reporting obfuscated employee locations, but these must be consistent with the building layout in order to not raise suspicion.

By expressing the requirement to hide their location as opacity, obfuscation techniques can be applied to the dynamic model induced by the building layout for each employee. However, the company also utilizes an emergency response service that must be informed when individuals are using the chemical lab. Existing methods of obfuscation cannot guarantee this form of utility as they view all recipients equally: if this information is hidden to competitors, it must be hidden from everyone. Instead, the company realizes they could provide information about their obfuscation to the emergency service, so that the service can infer the relevant information from the obfuscated locations. In summary, the company wants to design obfuscation and inference policies that are consistent with the dynamics induced by the building layout, guarantee privacy from their competitors, and maintain utility with the emergency service.

The rest of this work is organized as follows. In Section 2, we review results from formal languages and reactive synthesis. Next in Section 3, we formulate the problem of obfuscation synthesis with inference. We then present a solution to this problem by transforming it to a standard distributed synthesis problem in Section 4. Then in Section 5 we discuss our implementation of this method and present a case study of privacy for contact tracing using our approach.

2 PRELIMINARIES

In this section we review concepts from formal languages and automata theory and then techniques from reactive and distributed synthesis. A general introduction to these topics can be found in [1] and [4].

2.1 Formal Languages

For the finite alphabet Σ we denote the set of finite sequences, nonempty finite sequences, and infinite sequences over Σ by Σ^*, Σ^+, and Σ^ω, respectively. We denote the empty sequence as ϵ. A *language* is a subset $L \subseteq \Sigma$ while an *ω-language* is a subset $P \subseteq \Sigma^\omega$. For finite sequences, also called *words*, $s, s' \in \Sigma^*$, we write their concatenation as $ss' \in \Sigma^*$. We denote the set of finite *prefixes* of a word s, language L, infinite word t, and ω-language P by $\overline{s}, \overline{L}, \overline{t}, \overline{P} \subseteq \Sigma^*$, respectively. The *limit* of a language L is the ω-language $\lim L$ containing infinite words with infinitely many prefixes in L. We say an ω-language P is *closed*, also called a safety property, if $P = \lim \overline{P}$. Closed ω-languages P_1 and P_2 satisfy the following properties:

$$\overline{P_1} \cap \overline{P_2} = \overline{P_1 \cap P_2}, \qquad \overline{P_1} \subseteq \overline{P_2} \;\Leftrightarrow\; P_1 \subseteq P_2\,. \tag{1}$$

An *automaton* is a tuple $G = (Q, \Sigma, \delta, Q_0)$ with states Q, alphabet Σ, partial transition function $\delta : Q \times \Sigma \to 2^Q$, and initial states $Q_0 \subseteq Q$. A run of G over the word $t_0 \cdots t_{n-1} \in \Sigma^*$ is a sequence of states $q_0, \cdots, q_n \in Q$ such that $q_0 \in Q_0$ and for all $j < n$ it holds that $q_{j+1} \in \delta(q_j, t_j)$. The language generated by G denoted $\mathcal{L}(G) \subseteq \Sigma^*$ contains the words with a corresponding run over G. We say G is finite if Q and Σ are finite sets, while the size of G refers to the number of states $|Q|$. We say G is deterministic if $|Q_0| = 1$ and for all $q \in Q$ and $\sigma \in \Sigma$, it holds that $|\delta(q, \sigma)| = 1$. We can also consider infinite runs over automata along with acceptance conditions for runs. A Büchi automaton is an automaton augmented with a set of accepting states Q_f denoted by $H = (Q, \Sigma, \delta, Q_0, Q_f)$. An infinite run over an infinite word is accepted by H if it contains an element of Q_f infinitely often. The language accepted by H denoted by $\mathcal{L}(H)$ contains all infinite words with a corresponding accepting run over H. We say an ω-language is *ω-regular* if it is accepted by a finite Büchi automaton. In order to reason about finite behaviors using infinite ones we use the following simple result.

Lemma 2.1. *If $L = \mathcal{L}(G)$ where $G = (Q, \Sigma, \delta, Q_0)$ is finite and has no deadlocked state, i.e., each state has an outgoing transition, then $\lim(L)$ is closed and ω-regular, being accepted by the finite Büchi automaton $H = (Q, \Sigma, \delta, Q_0, Q)$, and $\overline{\lim L} = L$.*

2.2 Reactive Synthesis

We consider systems whose inputs and outputs are Boolean variables that evolve over time, i.e., for variables V we consider the alphabet $\Sigma = 2^V$. Given a sequence, also called a *trace*, $t = t_0 t_1 \cdots \subseteq \left(2^V\right)^\omega$, we define the *restriction* of t from V onto a subset of variables $U \subseteq V$ by $t|_U = (t_0 \cap U)(t_1 \cap U) \cdots$. We similarly define restrictions for finite sequences. A reactive *process* dynamically produces outputs based upon inputs it has received from the environment. Formally a process $\Psi = (I, O)$ is defined by a finite set of input variables I and output variables O. An implementation of a process Ψ is a *strategy* $\psi : \left(2^I\right)^+ \to 2^O$ that maps a nonempty finite sequence or *history* of inputs to the current output. Such strategies can be represented as a type of automaton called a *transducer*, defined over an alphabet consisting of the input and the output symbols. We call a strategy *finite* if it has a finite representation as a transducer.

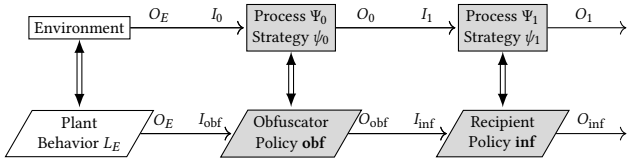

Figure 2: The pipeline architecture $\mathcal{A}$ (top) and the structure of the obfuscation system (bottom).

We model the obfuscation system as a distributed system where output from the plant (the environment) flows into the obfuscator process and then into the inference process at the recipient. This arrangement of processes is known as a *pipeline architecture* [10, 18] or one-way chain [15]. Formally, we define a pipeline architecture as a triple $\mathcal{A} = (O_E, \Psi_0, \Psi_1)$ where O_E are the outputs of the environment and $\Psi_0 = (I_0, O_0)$ and $\Psi_1 = (I_1, O_1)$ are controlled processes such that $I_0 \subseteq O_E$, $I_1 \subseteq O_0$, and O_E, O_0, O_1 are pairwise disjoint[1]. This architecture is depicted in Figure 2. Importantly, our definition allows for partial observation in that a process may not observe all outputs of the previous process. For the pipeline architecture $\mathcal{A}$ we define the set of variables $V(\mathcal{A}) = O_E \cup O_0 \cup O_1$. An implementation of $\mathcal{A}$ consists of strategies ψ_0 and ψ_1 for each of its processes. A trace of the implementation (ψ_0, ψ_1) is a trace over $V(\mathcal{A})$ that is consistent with the strategies ψ_0 and ψ_1. For a given architecture $\mathcal{A}$, the set of such traces is defined by

$$Tr(\psi_0, \psi_1) = \{t = t_0 t_1 \cdots \in \left(2^{V(\mathcal{A})}\right)^\omega \mid \forall j \in \{0,1\},\ \forall n \in \mathbb{N} :\ t_n|_{O_j} = \psi_j((t_0 \cdots t_n)|_{I_j})\}\,. \quad (2)$$

It is straightforward to show that the set of traces $Tr(\psi_0, \psi_1)$ is closed [22]. The distributed synthesis problem involves determining an implementation of a distributed architecture that satisfies a given specification. The synthesis method of [10] considers specifications φ over the computation tree of the implementation, for example given by μ-calculus [16]. In this work we consider a subclass of these specifications known as *linear-time* properties given by ω-regular languages describing traces of the system..

Problem 1 (Distributed Synthesis). *Given the pipeline architecture $\mathcal{A} = (O_E, \Psi_0, \Psi_1)$ and specification $\varphi \subseteq \left(2^{V(\mathcal{A})}\right)^\omega$, find strategies ψ_0 and ψ_1 implementing $\mathcal{A}$ such that*

$$\forall t \in Tr(\psi_0, \psi_1) :\ t \in \varphi\,. \quad (3)$$

It is well-known that the distributed synthesis problem is decidable for the pipeline architecture, and furthermore when a solution exists, there must also exist finite solutions [10, 18]. In this case for two processes, along with the environment, the size of the synthesized solutions are double-exponential in the size of the Büchi automaton accepting the specification.

Remark 1. *The works [10, 18] consider the distributed synthesis problem with delay, i.e., outputs from one process are available to the next process after one step. There is a simple transformation to the problem without delay we consider here.*

[1]Our notion of the pipeline architecture $\mathcal{A} = (O_E, \Psi_0, \Psi_1)$ is readily expressed as the more general notion of architectures defined in [10].

3 OBFUSCATION PROBLEM

In this section, we formulate the problem of obfuscation synthesis with formal models for the plant, obfuscator, and recipients. We express security and utility requirements through opacity and a new notion of inference.

3.1 System Model & Requirements

We consider a plant that sequentially outputs over a set of Boolean variables O_E. This behavior is defined by a set of finite traces $L_E \subseteq \left(2^{O_E}\right)^*$. The obfuscator **obf** receives a subset $I_{\text{obf}} \subseteq O_E$ of the plant's output variables as input and produces a sequence of outputs on the Boolean variables O_{obf} at each step. We only consider obfuscators that are *deterministic*, i.e., producing a single sequence of outputs on a single input, and *non-silent*, i.e., never producing the empty sequence. We call the implementation of an obfuscator an *obfuscation policy*.

Definition 3.1. *A deterministic and non-silent* obfuscation policy *is a function* $\mathbf{obf} : \left(2^{I_{obf}}\right)^+ \to \left(2^{O_{obf}}\right)^+$ *that maps input histories to a sequence of produced outputs.*

This definition is similar to the concept of an *edit functions* as in [14]. Given an input history $i = i_0 \cdots i_n \in \left(2^{I_{\text{obf}}}\right)^+$, the corresponding output histories consist of the complete output sequences made for i_0 through i_{n-1} followed by a partial output sequence made for i_n. We define the set of such *output histories* $\mathbf{Hist}(\mathbf{obf}, i)$ by

$$\mathbf{Hist}(\mathbf{obf}, i) = \{\mathbf{obf}(i_0) \cdots \mathbf{obf}(i_0 \cdots i_{n-1})\} \cdot (\overline{\mathbf{obf}(i_0 \cdots i_n)} \setminus \{\epsilon\})\,. \quad (4)$$

For example, if $\mathbf{obf}(i_0) = o_0 o_1$ and $\mathbf{obf}(i_0 i_1) = o_2 o_3$ then the output histories are $\mathbf{Hist}(\mathbf{obf}, i_0 i_1) = \{o_0 o_1 o_2,\ o_0 o_1 o_2 o_3\}$.

Recipients passively observe a subset $I_{\text{inf}} \subseteq O_{\text{obf}}$ of the obfuscator's output variables and try to reason about the state of the plant. Importantly we assume that recipients *only observe the outputs of the obfuscator sequentially, not how they are produced*, e.g. they cannot distinguish outputting the word $\sigma\sigma$ on one input and outputting σ twice over two inputs. This motivates our definition of the output histories from equation (4). Privacy and security requirements express limits on the knowledge deduced by unintended recipients. In the context of opacity [14], the notion of *private safety* describes privacy from a recipient that knows the plant but is unaware of obfuscation. In this case the recipient's observations should correspond to observations of nonsecret behavior in the plant. In this work, we more generally require these observations to belong to some *nonsecret* language L_{NS}.

Definition 3.2. *We say an obfuscation policy* **obf** *enforces* private safety *with respect to the plant behavior L_E and* nonsecret *output histories $L_{NS} \subseteq \left(2^{O_{obf}}\right)^*$ if*

$$\forall e \in L_E \setminus \{\epsilon\},\ \forall h \in \mathbf{Hist}(\mathbf{obf}, e|_{I_{obf}}) :\ h \in L_{NS}\,. \quad (5)$$

In the next subsection, we show how this notion of private safety can express the inability of a recipient with uncertain knowledge of the obfuscation policy to infer information about the plant. For instance, we construct L_{NS} for Example 1.1 as the set of location histories that do not visit the chemistry lab. In this case, private safety ensures that competitors can never deduce when employees

enter the chemistry lab. For simplicity, we will assume that both L_E and L_{NS} satisfy the conditions of Lemma 2.1 so that they can be generated by finite automata without deadlock.

While obfuscation ensures security against unintended recipients, the *intended recipient* must be able to *infer* certain information about the plant's behavior using the outputs of the obfuscator. So the inputs to the recipient are $I_{\text{inf}} = O_{\text{obf}}$ while their outputs O_{inf} encode their inferences about the plant with Boolean variables. The recipient's reasoning is then modeled by an *inference policy* $\mathbf{inf} : \left(2^{I_{\text{inf}}}\right)^+ \to 2^{O_{\text{inf}}}$ mapping the recipient's history of observations to their inferences. While the recipient could reason about arbitrary predicates over the plant behavior, we consider only their ability to always infer some fixed function $\mathbf{data} : 2^{O_E} \to 2^{O_{\text{inf}}}$ of the current plant output. This means if the current plant output is $e \in 2^{O_E}$, then the recipient's inference policy should output $\mathbf{data}(e)$. The number of possible inferences is given by $2^{|O_{\text{inf}}|}$. In the context of Example 1.1, the emergency service must infer when the user is in the chemistry lab, so we define **data** to output true when there is a user present and false otherwise.

Definition 3.3. *Given the plant behavior $L_E \subseteq \left(2^{O_E}\right)^*$, obfuscation policy $\mathbf{obf} : \left(2^{I_{obf}}\right)^+ \to \left(2^{O_{obf}}\right)^+$, and data function $\mathbf{data} : 2^{O_E} \to 2^{O_{inf}}$, an inference policy $\mathbf{inf} : \left(2^{I_{inf}}\right)^+ \to 2^{O_{inf}}$ is correct if*

$$\forall e = e_0 \cdots e_n \in L_E \setminus \{\epsilon\},\ \forall h \in \mathbf{Hist}(\mathbf{obf}, e|_{I_{obf}}) :\ \mathbf{inf}(h) = \mathbf{data}(e_n)\,. \tag{6}$$

With the models of the system defined along with the notions of correct inferences and private safety, we can now define the obfuscation synthesis problem.

Problem 2 (Obfuscation synthesis). *Given a plant with behavior L_E, nonsecret output histories L_{NS}, and information* **data**, *find an obfuscation policy* **obf** *that enforces private safety and an inference policy* **inf** *that is correct.*

After designing the obfuscation and inference policies, the inference policy can be securely transferred to the intended recipient. Critically, this allows the intended recipient to infer information about the plant that unintended recipients cannot.

3.2 Modeling Security Requirements

We now show how to construct the language L_{NS} to express security properties as private safety. In the context of opacity, reference [14] provides a definition of L_{NS} expressing the inability of an observer unaware of any obfuscation to deduce the plant is currently at a secret state. We extend this definition by considering recipients with some fixed but uncertain knowledge of the plant and obfuscation policy *a priori*. Namely they believe the plant behavior belongs to the set $L'_E \subseteq \left(2^{O_E}\right)^+$ and that this behavior is altered with an obfuscation policy in the class $\Theta \subseteq \{\mathbf{obf}' : \left(2^{I_{\text{obf}}}\right)^+ \to \left(2^{O_{\text{obf}}}\right)^+\}$. We consider the security requirement that the output of the obfuscator must be consistent with the recipient's model of L'_E and Θ. Additionally, the recipient must not be able to deduce that the plant behavior did not belong to some set of nonsecret behavior $L'_{E,\text{NS}} \subseteq L'_E$. We clarify that while the unintended recipient may know that their observations have been altered, we assume they do not know for what purpose or how the obfuscation is designed[2]. In this case it suffices to ensure any output history of the true obfuscation policy **obf** over the plant behaviors L_E is also the output history of some obfuscation policy $\mathbf{obf}' \in \Theta$ applied to a nonsecret plant trace $e'_{\text{NS}} \in L'_{E,\text{NS}}$. Formally,

$$\begin{aligned}&\forall e \in L_E \setminus \{\epsilon\},\ \forall h \in \mathbf{Hist}(\mathbf{obf}, e|_{I_{\text{obf}}}),\\ &\qquad \exists e'_{\text{NS}} \in L'_{E,\text{NS}},\ \exists \mathbf{obf}' \in \Theta :\ h' \in \mathbf{Hist}(\mathbf{obf}', e'_{\text{NS}}|_{I_{\text{obf}}})\,.\end{aligned} \tag{7}$$

A class of obfuscation policies $\Theta = \{\mathbf{obf}' : \left(2^{I_{\text{obf}}}\right)^+ \to \left(2^{O_{\text{obf}}}\right)^+\}$ defines a relation $\mathbf{R}_\Theta \subseteq \left(2^{I_{\text{obf}}}\right)^+ \times \left(2^{O_{\text{obf}}}\right)^+$ between plant behaviors and the corresponding possible output histories defined by

$$\mathbf{R}_\Theta = \bigcup_{e' \in \left(2^{I_{\text{obf}}}\right)^+} \bigcup_{\text{obf}' \in \Theta} \{e'\} \times \mathbf{Hist}(\mathbf{obf}', e')\,. \tag{8}$$

We call such a relation *regular* if it can be represented by a finite transducer, i.e., $\mathbf{R}_\Theta$ is the set of all pairs of input and output words accepted by the transducer. These relations are also called rational transductions in [20]. The *composition* $\mathbf{R}(L)$ of a relation $\mathbf{R}$ to a language L is defined by

$$\mathbf{R}(L) = \{h \mid \exists e \in L :\ (e, h) \in \mathbf{R}\}\,. \tag{9}$$

We assume that the behavior $L'_{E,\text{NS}}$ satisfies the conditions of Lemma 2.1 and the class of obfuscation policies Θ defines a *regular relation*. Under these assumptions, we have the following result.

Theorem 3.1. *Let $L'_{E,NS}$ be a language satisfying the conditions of Lemma 2.1 and let Θ be such that $\mathbf{R}_\Theta$ is regular. Then $L_{NS} = \mathbf{R}_\Theta(L'_{E,NS} \setminus \{\epsilon\})$ also satisfies the conditions of Lemma 2.1. Furthermore, an obfuscation policy* **obf** *is privately safe with respect to L_E and L_{NS} if and only if unintended recipients never deduce the plant behavior did not belong to $L'_{E,NS}$, i.e, condition* (7) *holds.*

Proof. By assumption, there exists a finite automaton G with $\mathcal{L}(G) = L'_{E,\text{NS}}$. Then as $\mathbf{R}_\Theta$ is a regular relation, by the results of [20] the composition $L_{\text{NS}} = \mathbf{R}_\Theta(L'_{E,\text{NS}} \setminus \{\epsilon\})$ can be represented by an automaton constructed as the product of the finite transducer representing $\mathbf{R}_\Theta$ with G. As both G and the transducer are both finite and deadlock-free (as obfuscation policies are defined over all inputs), it follows that this product automaton is also deadlock-free in the sense of Lemma 2.1. Hence L_{NS} satisfies the conditions of Lemma 2.1.

Next observe by the definition of L_{NS} and $\mathbf{R}_\Theta$ that

$$\begin{aligned}h \in L_{\text{NS}} &\overset{(9)}{\iff} \exists e' \in L'_{E,\text{NS}} :\ (e', h) \in \mathbf{R}_\Theta\\ &\overset{(8)}{\iff} \exists e' \in L'_{E,\text{NS}},\ \exists \mathbf{obf}' \in \Theta :\ h \in \mathbf{Hist}(\mathbf{obf}', e')\,.\end{aligned}$$

Hence condition (7) is equivalent to private safety. □

By explicitly modeling the knowledge of unintended recipients, we have control over the level of security the obfuscator guarantees. We also note that the more uncertain the unintended recipient, i.e., the larger $L'_{E,\text{NS}}$ and Θ are, the easier it is to enforce private safety.

[2] In future work, we can consider obfuscators that are secure to recipients aware of the design requirements and synthesis method. This would be similar to the notion of *public safety* [14].

For example if the unintended recipient is not aware that the plant outputs are altered, we consider the class Θ with only the identity map. This corresponds to the existing notion of private safety in [13]. If instead the unintended recipient is more uncertain and believes that the outputs could be altered but there can only be k consecutive outputs for a single input, we consider the class $\Theta = \{\mathbf{obf} \mid \forall e : |\mathbf{obf}(e)| \leq k\}$. In this case, any solution for the first case will also be a solution for the second case. In both bases, we can also see that the induced relations $\mathbf{R}_\Theta$ are regular.

Example 3.1. *We transform the problem described in Example 1.1 into an instance of Problem 2 as follows. First we model the movement of the employee around the building layout graph Γ from Figure 1 and the regions detected by the smart devices as the plant behavior L_E. We define this behavior over the possible regions and whether or not the current location is the secret one (the chemistry lab). So we define $O_E = \{\mathcal{R}, \mathcal{T}, \mathcal{U}, \mathrm{S}\}$ where S denotes the secret*[3]*. We construct an automaton that generates the behavior L_E with states given by the locations in Γ. Then for each movement from one location to another given by an edge in Γ, we add a transition labeled with the region and secret status of the destination. This results in the automaton G depicted in Figure 3.*

Next, we model the obfuscator. Recall the obfuscator only observes the current region (and not the secret output S) so we have $I_{obf} = \{\mathcal{R}, \mathcal{T}, \mathcal{U}\}$. Likewise the obfuscator outputs regions which we represent with copies of the region variables $O_{obf} = \{\mathcal{R}', \mathcal{T}', \mathcal{U}'\}$ (variables across processes must be disjoint). We can then express the privacy requirement as private safety. Recall the unintended recipients are competitors that should not be able to deduce that an employee is in the chemistry lab. As the competitor knows the true layout of the building, their model of the plant is also given by $L'_E = \mathcal{L}(G)$. The nonsecret behavior $L'_{E,NS}$ is then given by the language generated by G after removing the secret state, i.e., $L'_{E,NS} = \overline{\{\mathcal{U}\mathcal{R}, \mathcal{T}\mathcal{R}\}^*}$. As we assume the competitor will not be aware of this obfuscation, we consider the class Θ consisting only of the identity map (mapping regions to their copy). We can then construct $L_{NS} = \mathbf{R}_\Theta(L'_{E,NS}) = \overline{\{\mathcal{U}'\mathcal{R}', \mathcal{T}'\mathcal{R}'\}^*}$ as in Theorem 3.1 to define the appropriate notion of private safety.

Finally, we model the intended recipient. Recall the intended recipient is the emergency service that observes outputs from the obfuscator and must be able to infer when an employee is in the chemistry lab. So we define $I_{inf} = O_{obf}$ and $O_{inf} = \{\mathrm{S}'\}$ and $\mathbf{data} : 2^{O_E} \to 2^{O_{inf}}$ with $\mathbf{data}(e) = \{\mathrm{S}'\}$ if $\mathrm{S} \in e$ and $\mathbf{data}(e) = \emptyset$ otherwise. Here S' is a copy of the plant variable S that is true when the recipient infers an employee is in the chemistry lab.

The company then desires to solve Problem 2 to design an obfuscation policy **obf** *and inference policy* **inf**. *By having each employee obfuscate their location with* **obf** *on their smart device, they ensure their competitors will not think they visit the chemistry lab. Then by securely distributing* **inf** *to the emergency service, they ensure they will be able to know when they visit the chemistry lab.*

4 OBFUSCATION SYNTHESIS

In this section, we show how to transform Problem 2 for obfuscation synthesis into an instance of Problem 1 for pipeline synthesis that we know how to solve. While the system of Problem 2 resembles the pipeline architecture of Problem 1 with the plant feeding into the obfuscation policy feeding into the inference policy, it is not "synchronous": the obfuscation policy produces a variable-length sequence of outputs on consuming a single input. To address this issue we *unfold* the obfuscation system and specifications for private safety and correct inferences into synchronous ones. Then solutions to Problem 2 can be found by folding solutions found to an instance of Problem 1 over the unfolded system.

[3]For simplicity, we encode each region with a single variable. As they are disjoint, they could more efficiently be encoded with two variables rather than three total.

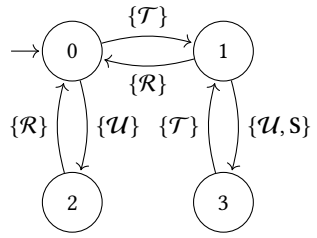

Figure 3: An automaton G generating the plant behavior L_E corresponding to the employee's movement throughout the building depicted in Figure 1. Each region of the building is encoded with its own variable $\mathcal{R}, \mathcal{T}, \mathcal{U}$ along with the secret status of the room encoded with the variable S .

4.1 Unfolding the System

We must *unfold* our the obfuscation system so that one output is produced by the obfuscation policy in each step. This unfolded system has the same plant outputs O_E as the original or *folded* system. We represent the obfuscator with a process $\Psi_0 = (I_0, O_0)$ with the same inputs $I_0 = I_{\text{obf}}$ but with outputs $O_0 = O_{\text{obf}} \cup \{\text{yield}\}$ augmented with a variable yield indicating the sequence of outputs has completed. As the inference with an inference policy is already synchronous, we can represent it directly with the process $\Psi_1 = (I_1, O_1)$ with $I_1 = I_{\text{inf}}$ and $O_0 = O_{\text{inf}}$. Here the yield variable is output by the obfuscator process Ψ_0 because the obfuscation policy controls the length of its output sequences; however, the yield variable is not available to the inference process Ψ_1 as input because recipients do not observe these lengths. This defines a pipeline architecture $\mathcal{A} = (O_E, \Psi_0, \Psi_1)$. We now show how to unfold behavior of the original or *folded* system into behavior over $\mathcal{A}$.

In a single step of the folded system, the plant generates an output $v \in 2^{O_E}$, the obfuscation policy outputs the sequence $o_0 \cdots o_n \in \left(2^{O_E}\right)^+$, and finally the inference policy makes a corresponding sequence of inferences $p_0 \cdots p_n \in \left(2^{O_{\text{inf}}}\right)^+$. By defining $f = (v, (o_0 \cup p_0) \cdots (o_n \cup p_n))$ we can view a step of the folded system as elements of the set

$$F = 2^{O_E} \times \left(2^{O_{\text{obf}} \cup O_{\text{inf}}}\right)^+. \tag{10}$$

To unfold this step of the original system, we break each output of the obfuscation policy into a single step. The auxiliary output yield is used to indicate that output has completed. As there must be one environment output each step, we have it output v on the first step followed by $\emptyset$ on subsequent steps. So we unfold the step,

as $\mathbf{unfold}(f) = t_0 \cdots t_n \in \left(2^{V(\mathcal{A})}\right)^+$ where

$$t_0 = v \cup o_0 \cup p_0, \qquad \forall j \in \{1, \cdots, n-1\} : \ t_j = o_j \cup p_j, \tag{11}$$

$$t_n = o_n \cup p_n \cup \{\text{yield}\}, \tag{12}$$

when $n > 0$ and $t_0 = v \cup o_0 \cup p_0 \cup \{\text{yield}\}$ when $n = 0$.

Now we extend this notion from a single step to infinite traces. To this end, we denote the flattening or concatenating of a sequence of empty words into a sequence of letters by $\mathbf{flat} : (\Sigma^+)^\omega \to \Sigma^\omega$ so that for $s = (s_j)_{j\in\mathbb{N}}$ with $s_j \in \Sigma^+$, it holds $\mathbf{flat}(s) = s_0 s_1 \cdots$. We can then define the *traces* of the folded system by

$$\begin{aligned} Tr(\mathbf{obf}, \mathbf{inf}) = \{(e_j, a_j)_{j=0}^\infty \in F^\omega \mid \\ \forall j \in \mathbb{N} : \ a_j|_{O_{\text{obf}}} = \mathbf{obf}(e_0 \cdots e_j|_{I_{\text{obf}}}), \\ \text{for } (\tilde{a}_k)_{k\in\mathbb{N}} = \mathbf{flat}((a_j)_{j\in\mathbb{N}}), \\ \forall k \in \mathbb{N} : \ \tilde{a}_k|_{O_{\text{inf}}} = \mathbf{inf}(\tilde{a}_0 \cdots \tilde{a}_k|_{I_{\text{inf}}})\} . \end{aligned} \tag{13}$$

We denote the set of words possible after unfolding a single step by $U = \mathbf{unfold}(F) \subseteq \left(2^{V(\mathcal{A})}\right)^+$. So to unfold a trace in F^ω, we unfold each each step to an element of U and flatten the result. Formally, we define

$$P_U = \mathbf{flat}(U^\omega) \subseteq \left(2^{V(\mathcal{A})}\right)^\omega, \tag{14}$$

and $\mathbf{unfold} : F^\omega \to P_U$ by

$$\mathbf{unfold}(f_0 f_1 \cdots) = \mathbf{unfold}(f_0)\mathbf{unfold}(f_1) \cdots . \tag{15}$$

To demonstrate unfolding, consider a behavior of the plant from Example 3.1 as depicted in Figure 3 where the employee moves from the lobby to the office to the chemistry lab, i.e., $\{\mathcal{T}\} \to \{\mathcal{U}, \mathrm{S}\}$. We consider the obfuscation and inference policies **obf** and inf depicted in Figure 4. Over this path, the obfuscator first outputs movement to office and back to the lobby, i.e., $\{\mathcal{T}'\} \to \{\mathcal{R}'\}$, then movement to the electronics lab and back to the lobby $\{\mathcal{U}'\} \to \{\mathcal{R}'\}$. We see as the inference policy is correct, for the first two locations it infers that they are not in the chemistry lab, i.e., $\emptyset$, then on the next two that they are, i.e., $\{\mathrm{S}'\}$. This corresponds to the finite folded trace

$$f = (\underbrace{\{\mathcal{T}\}}_{e}, \underbrace{\{\mathcal{T}'\}}_{a_0} \underbrace{\{\mathcal{R}'\}}_{a_1}) \ (\{\mathcal{U}, \mathrm{S}\}, \{\mathcal{U}', \mathrm{S}'\}\{\mathcal{R}', \mathrm{S}'\}) .$$

This is unfolded to

$$\mathbf{unfold}(f) = \underbrace{\{\mathcal{T}, \mathcal{T}'\}}_{t_0} \underbrace{\{\mathcal{R}', \text{yield}\}}_{t_1} \{\mathcal{U}, \mathrm{S}, \mathcal{U}', \mathrm{S}'\}\{\mathcal{R}', \mathrm{S}', \text{yield}\} .$$

The following result shows that **unfold** can be inverted to *fold* traces. Importantly, this defines a transformation between behaviors in the folded setting of Problem 2 and unfolded setting of Problem 1.

Lemma 4.1. *The map* $\mathbf{unfold} : F^\omega \to P_U$ *is a bijection.*

Proof. From the definition for a single step, we see **unfold** defines a bijection between F and U. Then as $\mathbf{unfold}(f_0 f_1 \cdots) = \mathbf{unfold}(f_0)\mathbf{unfold}(f_1) \cdots$ we see **unfold** is surjective onto P_U. Finally as every word in U ends with yield, there is a unique way to write traces in P_U as sequences of words in U (delimited by yield). So as **unfold** is injective for a single step, **unfold** is injective over F^ω. Hence **unfold** is a bijection. □

In addition to unfolding the traces of an obfuscation policy $\mathbf{obf} : \left(2^{I_{\text{obf}}}\right)^+ \to \left(2^{O_{\text{obf}}}\right)^+$ and inference policy $\mathbf{inf} : \left(2^{I_{\text{inf}}}\right)^+ \to 2^{O_{\text{inf}}}$, we can also unfold the functions themselves into strategies $\psi_0 : \left(2^{I_0}\right)^+ \to 2^{O_0}$ and $\psi_1 : \left(2^{I_1}\right)^+ \to 2^{O_1}$ implementing the architecture $\mathcal{A}$. Note that strategies are defined for all input sequences, even those violating the yield behavior encoded in P_U. These violating traces do not correspond to traces in the folded system. As such, we say the strategies ψ_0 and ψ_1 are an *unfolding* of **obf** and **inf** if

$$\mathbf{unfold}(Tr(\mathbf{obf}, \mathbf{inf})) = Tr(\psi_0, \psi_1) \cap P_U . \tag{16}$$

As obfuscation policies can only insert a finite sequence of outputs, we should only consider strategies ψ_0 that *always eventually yield*, i.e.,

$$\forall i \in \left(2^{I_0}\right)^+, \ \exists v \left(2^{I_0}\right)^+ : \ \text{yield} \in \psi_0(iv) . \tag{17}$$

In this case we have the following results.

Theorem 4.2.
***Unfolding**: Every obfuscation policy* **obf** *and inference policy* **inf** *has an unfolding in the sense of* (16) *given by strategies* ψ_0 *and* ψ_1 *where* ψ_0 *always eventually yields and* $\psi_1 = \mathbf{inf}$.
***Folding**: For every strategy* ψ_0 *that always eventually yields and strategy* ψ_1*, there exists a unique obfuscation policy* **obf** *and inference policy* **inf** *such that* ψ_0 *and* ψ_1 *are an unfolding in the sense of* (16) *of* **obf** *and* **inf** *with* $\psi_1 = \mathbf{inf}$.

Proof. See appendix. □

This result shows that we can transform between policies over the folded system and strategies implementing the unfolded distributed pipeline architecture $\mathcal{A}$ as in Figure 2. Similar to strategies, the obfuscation policy can be represented by a transducer, and when this transducer is finite we say the policy is finite. As unfolding and folding preserve ω-regularity, this theorem implies that a solution to Problem 2 is finite if and only if it has an unfolding that is finite.

4.2 Unfolding the Specifications

In order to perform distributed synthesis on the unfolded system, we must map specifications for the folded system onto the unfolded one. To do this, we observe that private safety of the obfuscation policy **obf** and the correctness of the inference policy **inf** can be expressed as properties over the traces $Tr(\mathbf{obf}, \mathbf{inf})$. Then by unfolding these traces, we can express these requirements as ω-regular properties over the traces $Tr(\psi_0, \psi_1)$ where ψ_0 and ψ_1 are an unfolding of **obf** and **inf**. We define the properties of the folded system representing the plant behavior, private safety, and correct inferences, respectively, by

$$\begin{aligned} P_E &= \{(e_j, a_j)_{j=0}^\infty \in F^\omega \mid e_0 e_1 \cdots \in \lim L_E\} \\ P_{\text{NS}} &= \{(e_j, a_j)_{j=0}^\infty \in F^\omega \mid a_0 a_1 \cdots|_{O_{\text{obf}}} \in \lim L_{\text{NS}}\} \\ P_{\text{inf}} &= \{(e_j, a_j)_{j=0}^\infty \in F^\omega \mid \forall j \in \mathbb{N} : \ a_j|_{O_{\text{inf}}} = \underbrace{\mathbf{data}(e_j) \cdots \mathbf{data}(e_j)}_{|a_j| \text{ times}}\} . \end{aligned}$$

Next, we construct Büchi automata that accept the unfolding of each of these properties.

To unfold the plant behavior, we recall that in the unfolded system the plant only progresses after the outputs have yielded and

outputs $\emptyset$ otherwise. As L_E satisfies the conditions of Lemma 2.1, there exists a finite Büchi automaton $H_E = (Q_E, 2^{O_E}, \delta_E, Q_{E,0}, Q_E)$ accepting $\lim L_E$. Let $H'_E = (Q'_E, 2^{V(\mathcal{A})}, \delta'_E, Q'_{E,0}, Q'_{E,m})$ where $Q'_E = Q_E \times \{0,1\}$, $Q'_{E,0} = Q_{E,0} \times \{0\}$, $Q'_{E,m} = Q_E \times \{0\}$, and

$$\delta'_E((q,b),v) = \begin{cases} \delta_E(q, v|_{O_E}) \times \{0\}, & b = 0 \wedge \text{yield} \in v \\ \delta_E(q, v|_{O_E}) \times \{1\}, & b = 0 \wedge \text{yield} \notin v \\ \{(q,0)\}, & b = 1 \wedge \text{yield} \in v \wedge v|_{O_E} = \emptyset \\ \{(q,1)\}, & b = 1 \wedge \text{yield} \notin v \wedge v|_{O_E} = \emptyset \\ \emptyset, & \text{otherwise} \end{cases}$$

The first component q of the states of H'_E follows a stuttered path of the plant automaton H_E, repeating the current plant state until yield has occurred. This occurrence is tracked by the second component b of the states of H'_E, where $b = 0$ indicates that yield has occurred and the plant can transition. Only the states with $b = 0$ are accepting as the obfuscation policy only outputs finite sequences, i.e., yield occurs infinitely often. By construction $\textbf{unfold}(P_E) = \mathcal{L}(H'_E)$. Furthermore, if H_E is deterministic then H'_E is as well and the size of H'_E is polynomial in the size of H_E.

Next, we unfold behavior representing private safety. As L_{NS} satisfies the conditions of Lemma 2.1, there exists a finite Büchi automaton

$$H_{\text{NS}} = (Q_{\text{NS}}, 2^{O_{\text{obf}}}, \delta_{\text{NS}}, Q_{\text{NS},0}, Q_{\text{NS}}) \tag{18}$$

accepting $\lim L_{\text{NS}}$. Let $H'_{\text{NS}} = (Q'_{\text{NS}}, 2^{V(\mathcal{A})}, \delta'_{\text{NS}}, Q'_{\text{NS},0}, Q'_{\text{NS},m})$ where $Q'_{\text{NS}} = Q_{\text{NS}}$, $Q'_{\text{NS},0} = Q_{\text{NS},0}$, $Q'_{\text{NS},m} = Q_{\text{NS}}$, and

$$\delta'_{\text{NS}}(q,v) = \delta_{\text{NS}}(q, v|_{O_{\text{obf}}}) . \tag{19}$$

The automaton H'_{NS} simply accepts traces of the unfolded system whose restriction to the obfuscation outputs O_{obf} are in L_{NS}. As unfolding does not alter these outputs, it holds that $\textbf{unfold}(P_{\text{NS}}) = P_U \cap \mathcal{L}(H'_{\text{NS}})$. Also, clearly H'_{NS} has the same number of states as H_{NS}.

Finally, we unfold behavior representing correct inferences, i.e, the inferred output is equal to the **data** function of the current plant output. In the unfolded system, the "current" plant output corresponds to the value of the variables O_E after the most recent yield (or the initial value). So we construct H'_{inf} accepting these traces as follows. Let $H'_{\text{inf}} = (Q'_{\text{inf}}, 2^{V(\mathcal{A})}, \delta'_{\text{inf}}, Q'_{\text{inf},0}, Q'_{\text{inf},m})$ where $Q'_{\text{inf}} = \{q_0\} \cup 2^{O_{\text{inf}}}$, $Q'_{\text{inf},0} = \{q_0\}$, $Q'_{\text{inf},m} = Q'_{\text{inf}}$, and

$$\delta'_{\text{inf}}(q,v) = \begin{cases} v|_{O_{\text{inf}}}, & \text{yield} \notin v \ \wedge \ v|_{O_{\text{inf}}} = q \\ v|_{O_{\text{inf}}}, & \text{yield} \notin v \ \wedge \ q = q_0 \ \wedge v|_{O_{\text{inf}}} = \textbf{data}(v|_{O_{\text{inf}}}) \\ q_0, & \text{yield} \in v \ \wedge \ v|_{O_{\text{inf}}} = q \\ q_0, & \text{yield} \in v \ \wedge \ q = q_0 \ \wedge v|_{O_{\text{inf}}} = \textbf{data}(v|_{O_{\text{inf}}}) \end{cases}$$

The state of automaton H'_{inf} tracks the "current" plant output until yield, and the automaton accepts only traces with inferences matching this output. Then it holds that $\textbf{unfold}(P_{\text{inf}}) = P_U \cap \mathcal{L}(H'_{\text{inf}})$. Also we note that the size of H'_{inf} is polynomial in the number of possible inferences.

Additionally, we express the requirement of finite output sequences, represented by always eventually yielding as in (17) in the unfolded system, with a finite Büchi automaton H'_{yield}. Let $H'_{\text{yield}} = (Q'_{\text{yield}}, 2^{V(\mathcal{A})}, \delta'_{\text{yield}}, Q'_{\text{yield},0}, Q'_{\text{yield},m})$ where $Q'_{\text{yield}} = \{0,1\}$, $Q'_{\text{yield},0} = \{0\}$, $Q'_{\text{yield},m} = \{0\}$ and

$$\delta'_{\text{yield}}(q,v) = \begin{cases} 0, & \text{yield} \in v \\ 1, & \text{yield} \notin v . \end{cases} \tag{20}$$

Then

$$\mathcal{L}(H'_{\text{yield}}) = \{t \in \left(2^{V(\mathcal{A})}\right)^{\omega} \mid \forall j \in \mathbb{N}, \ \exists k \geq j : \ \text{yield} \in t_k\} . \tag{21}$$

Also note H'_{yield} has two states. We combine these properties to create an specification capturing the desired behavior of the unfolded system

$$\varphi = \mathcal{L}(H'_{\text{yield}}) \cap (\mathcal{L}(H'_E)^c \cup (\mathcal{L}(H'_{\text{NS}}) \cap \mathcal{L}(H'_{\text{inf}}))) . \tag{22}$$

Using standard constructions for the union, product, and complement of Büchi automata [1], we can construct a finite Büchi automaton that accepts this specification. Assuming the automaton generating the plant behavior L_E is deterministic, the size of the specification automaton is polynomial in the size of the automata generating the plant behavior L_E, nonsecret behavior L_{NS}, and number of possible inferences. With this specification we present our main results.

Theorem 4.3. *Consider an obfuscation policy* **obf** *and inference policy* **inf** *with an unfolding given by ψ_0 and ψ_1 where ψ_0 always eventually yields in the sense of* (17). *Then* **obf** *and* **inf** *are solutions to Problem 2 with respect to L_E, L_{NS}, and* **data** *if and only if ψ_0 and ψ_1 are solutions to Problem 1 for the architecture $\mathcal{A}$ and specification φ.*

Proof. From Definitions 3.3 and 3.2, we see that **obf** enforces private safety and **inf** is correct with respect to L_E, L_{NS}, and **data** if and only if

$$\overline{Tr(\textbf{obf}, \textbf{inf})} \cap \overline{P_E} \subseteq \overline{P_{\text{inf}}} \cap \overline{P_{\text{NS}}} . \tag{23}$$

As $Tr(\textbf{obf}, \textbf{inf})$, P_E, P_{inf}, and P_{NS} are all closed, equation (1) shows this is equivalent to the infinite trace inclusion

$$Tr(\textbf{obf}, \textbf{inf}) \cap P_E \subseteq P_{\text{inf}} \cap P_{\text{NS}} . \tag{24}$$

As **unfold** is a bijection, we can unfold each side of this inclusion. So by assumption as $\textbf{unfold}(Tr(\textbf{obf}, \textbf{inf})) = P_U \cap Tr(\psi_0, \psi_1)$, the inclusion is equivalent to

$$Tr(\psi_0, \psi_1) \cap P_U \cap \textbf{unfold}(P_E) \subseteq \textbf{unfold}(P_{\text{NS}}) \cap \textbf{unfold}(P_{\text{inf}}) . \tag{25}$$

In turn by the definitions of $H_E, H_{\text{NS}}, H_{\text{inf}}$, this inclusion is equivalent to

$$Tr(\psi_0, \psi_1) \cap P_U \cap \mathcal{L}(H_E) \subseteq P_U \cap \mathcal{L}(H_{\text{NS}}) \cap \mathcal{L}(H_{\text{inf}}) . \tag{26}$$

Rearranging terms and using the fact that $\mathcal{L}(H_E) \subseteq P_U$, this is equivalent to

$$Tr(\psi_0, \psi_1) \subseteq \mathcal{L}(H_{\text{inf}}) \cap \mathcal{L}(H_{\text{NS}}) \cup \mathcal{L}(H_E)^c . \tag{27}$$

By assumption as ψ_0 always eventually yields, it must be that $Tr(\psi_0, \psi_1) \subseteq \mathcal{L}(H_{\text{yield}})$. Hence by the definition of φ, the inclusion is equivalent to $Tr(\psi_0, \psi_1) \subseteq \varphi$. □

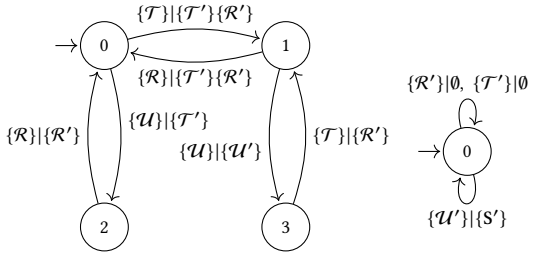

Figure 4: The solution to the obfuscation problem described in Example 3.1 given by transducers representing the obfuscation policy (left) and inference policy (right).

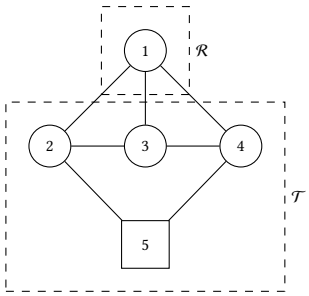

Figure 5: The graph Γ that represents the locations and the physical paths between them. Regions $P = \{\mathcal{R}, \mathcal{T}\}$ represent the approximate locations reported by smart phones. Location 5 is considered to be secret.

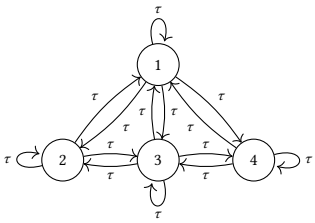

Figure 6: The automaton $\mathcal{G}_1$, representing the movement of the malicious user 1.

Applying the results of Theorem 4.2, we see if there is a solution to Problem 2 then there must be an unfolding that is a solution to Problem 1. Conversely, if there is a solution to Problem 1 the synthesis method [10] finds a finite solution. Applying the results of Theorem 4.2 that this solution can be folded into a solution to Problem 2 that is also finite. Hence if the obfuscation synthesis problem has a solution, finite obfuscation and inference policies solving the problem can be found by solving the corresponding pipeline synthesis problem and folding the result. Applying this approach to the problem from Example 3.1 yields the policies depicted in Figure 4. Assuming the automaton generating the plant behavior L_E is deterministic, the size of this solution is double-exponential in the size of the automata generating L_E and L_{NS} and number of possible inferences.

5 CASE STUDY: CONTACT TRACING

In this section, we demonstrate how our framework can be used in the context of smartphone apps developed for contact tracing. We model apps that record proximity between users to a centralized server. These apps raise a variety of privacy concerns as described in [3, 19], including the disclosure of user location information due to unsecured networks. Our synthesis method provides solutions that enforce location privacy from the malicious actors while maintaining utility for public health by providing professionals with relevant contact information. We demonstrate this approach on small model similar to the one developed in [24] where a malicious user has gained access to the app. Our presentation of this model is condensed. More detail on the construction can be found in [24].

5.1 Modeling

In our model, we consider a number of normal users and one user that is malicious. As in Example 1.1, their movement is constrained by a graph $\Gamma = (\mathcal{V}, \mathcal{E})$ depicted in Figure 5 representing the layout of their city. The users' smart devices would report their approximate location given by their current region in a partition P of the graph Γ similar to the model proposed in [26]. As such, we define **region** : $\mathcal{V} \to P$ so that for $v \in \mathcal{V}$, **region**(v) is the region containing v. We assume the malicious user has compromised the users' location-based service apps and has access to the reported regions, and also knows their own location exactly. We further assume that this user has compromised the contact tracing app which reports which users are in contact with each other, i.e., share the same location. As a privacy measure, the app does not report the location of the contact.

We suppose that there are $n = 3$ users and consider any user visiting the secret location 5 as secret in the plant. Using Γ, we first construct the map automaton $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Delta, \mathcal{V}_0)$, where $\mathcal{V}$ is the set of states, $\mathcal{E}$ is the set of events, $\Delta : \mathcal{V} \times \mathcal{E} \to 2^{\mathcal{V}}$ is the transition function, and $\mathcal{V}_0$ is the set of initial states. We define $\mathcal{V}_0 = \mathcal{V}$, $\mathcal{E} = \{\tau\}$, and $\Delta(v, \tau) = \{v\} \cup \{u \in \mathcal{V} : (u, v) \in \mathscr{E}\}$ for all $v \in \mathcal{V}$.For each user denoted $i \in \{1, 2, \ldots, n\}$, we construct an automaton $\mathcal{G}_i$ modeling their movement. We let user 1 denote the malicious user and construct $\mathcal{G}_1$ from $\mathcal{G}$ by removing the secret location (if the malicious user can enter the secret location, the problem trivially has no solution). For users $i > 1$, we let $\mathcal{G}_i = \mathcal{G}$ as their movements are unrestricted. For example, the individual automaton $\mathcal{G}_1$ is shown in Figure 6. The synchronous movement of all the users is then described by the product automaton $\mathcal{H} = \mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_n$. Using the product construction, $\mathcal{H}$ has states $x = (v_1, ..., v_n) \in \mathcal{V}^n$ where v_i represents the current location of user i.

Given a state $x \in \mathcal{V}^n$, we define

$$\alpha(x) = (\alpha_1, ..., \alpha_n) \tag{28}$$

where for each i,

$$\alpha_i = \begin{cases} v_1, & i = 1 \\ \textbf{region}(v_i), & i > 1. \end{cases} \tag{29}$$

When the system enters state x, the malicious user observes the information in $\alpha(x)$, i.e, their own location and the regions of other

users. We also define the set

$$\beta(x) = \{(i, j) \mid 1 \leq i < j \leq n, \;\; v_i = v_j\}, \tag{30}$$

which represents the contact information of the users at state x, i.e. the pairs of users that share the same location. To put this model into the setting of synthesis, we consider Boolean encodings of the outputs of α and β denoted by α_b and β_b, respectively. Finally, we construct an automaton G_E from $\mathcal{H}$ by labeling each transition to a state x with the variables $\alpha_b(x) \cup \beta_b(x)$. In order to construct the specification φ, we also determinize G_E using the powerset construction. The automaton G_E generates the plant behavior $L_E = \mathcal{L}(G_E)$ which satisfies the conditions of Lemma 2.1, i.e., G_E is finite and has no deadlocked states. Recall for privacy, the malicious user must not determine users are at the secret location, i.e., if in the current state of G_E any user is in the secret location which we call a secret state. As such we define the malicious user's nonsecret plant model $L'_{E,\mathrm{NS}}$ as the language generated by the automaton G_E after removal of the secret states. As we assume they are not aware of obfuscation, their class of possible obfuscation policies Θ is just the identity map. Finally, as the contact information represented by β_b should be inferred, we define the function **data** for the plant output $e = \alpha(x)_b \cup \beta_b(x)$ by $\mathbf{data}(e) = \tilde{\beta}_b(x)$, where $\tilde{\beta}_b(x)$ is a copy of the variables in $\beta_b(x)$ (to ensure variables are disjoint). Together these components define an instance of Problem 2 which by Theorem 4.3 can be transformed into a corresponding instance of Problem 1. In this form, we also add the additional ω-regular constraint that the obfuscation policy cannot alter the location of the malicious user encoded in α_b as this would alert them to the existence of obfuscation.

5.2 Implementation and Results

While the pipeline synthesis problem can be solved directly using automata theoretic methods, due to a lack of available tools and to take advantage of the performance of bounded synthesis methods, we use a different approach. There is a straightforward reduction of a distributed synthesis problem to a decidable hyperproperty satisfiability problem [8, 9]. Hyperproperties are properties of a system quantified over multiple traces of the system. The key idea of the reduction is to encode the variable dependence induced by the distributed architecture into a hyperproperty. Specifically, this hyperproperty ensures that any traces of the system with the same input up to a point must have the same output. Using this reformulation, we have implemented our synthesis method using the bounded synthesis tool *BoSy* [4]. We construct the Büchi automaton accepting the specification φ from (22), and perform minimization to reduce the number of states of this automaton while maintaining the specification language in order to improve performance. We then provide the tool with this automaton as well as the pipeline architecture encoded in HyperLTL, a temporal logic for hyperproperties. When a solution exists, the tool returns the smallest solution encoded as a finite automaton, otherwise the tool asserts unrealizability. Again, strategies for the obfuscation policy and the inference policy represented as transducers are readily extracted from this monolithic automaton. The implementation of this synthesis method is available in the *M-DESops* library[5].

The automaton G_E representing the plant behavior consists of 60 states after minimization. From this, we construct the Büchi automaton accepting the specification φ which has 968 states. With this automaton as input, the tool was able to synthesize a solution within 25 minutes on a machine with typical specs. As an automaton encoding both the obfuscation and inference policies, the solution mirrors the structure of the plant automaton G_E, possessing a corresponding 60 states. After extraction from the solution, the obfuscation and inference policies constructed guarantee privacy in the form of private safety while maintaining utility in the form of providing correct contact information.

6 CONCLUSION

Balancing privacy and utility within a networked dynamic system presents an interesting challenge. To achieve this, we propose a framework of obfuscation with an inference policy that allows intended recipients to interpret obfuscated information. We present a method for automatically designing both obfuscation and inference policies using techniques from distributed synthesis. This approach allows for a variety of specifications for utility and privacy in the form of temporal logic. We also developed a software implementation of this approach and demonstrate its effectiveness in enforcing privacy on a contact-tracing system model.

We remark that our proposed obfuscation framework is orthogonal to cryptographic methods for network privacy. While cryptography achieves privacy by ensuring outputs appear arbitrary, obfuscation achieves privacy by deception, i.e., ensuring outputs are consistent with a recipient's model of nonsecret behavior. In this way our obfuscation framework is similar to the notion of *network steganography* [17]. Additionally, our approach can be applied in cases where encryption cannot. While in this work we assumed the ability to output arbitrary messages, this approach could be applied in the setting where outputs are subject to additional dynamic constraints.

There are many directions for future work using this framework. For example, in this work we consider current-state opacity to hide a user's current location. More general notions like in [5] or K-step opacity [21] consider secrets that are time-sensitive. Dual to this, one might also consider requirements to infer information within a given number of steps. These notions may be expressed in temporal logic in our framework. Additionally, while in this work we consider a single intended recipient on the network, systems may have multiple recipients who need to infer heterogeneous domains of information. In this case multiple inference policies must be designed and must not reveal information outside of the recipients domain. Finally, while we consider unintended recipients that do not know how the obfuscation is designed, systems may require that security hold even when the synthesis method is public knowledge.

[4] https://www.react.uni-saarland.de/tools/bosy/

[5] https://gitlab.eecs.umich.edu/M-DES-tools/desops

ACKNOWLEDGMENTS

This work was supported in part by NSF grants CNS-1738103, CNS-1801342, and ECCS-1553873. Additional support was provided by a sponsored research award by Cisco Research.

REFERENCES

[1] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. The MIT Press, Cambridge, Mass.

[2] Henri-Charles Blondeel. 2008. Security by Logic : Characterizing Non-Interference in Temporal Logic. /paper/Security-by-logic-%3A-characterizing-Non-Interference-Blondeel/962e6ee9b2fddcdcbb55fd0b5296d9bd7fec9b40.

[3] Justin Chan, Landon Cox, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob Sunshine, and Stefano Tessaro. 2020. PACT: Privacy-Sensitive Protocols and Mechanisms for Mobile Contact Tracing. *IEEE Data Engineering Bulletin* 43, 2 (July 2020), 15–35.

[4] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). 2018. *Handbook of Model Checking*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-10575-8

[5] Rayna Dimitrova, Bernd Finkbeiner, Máté Kovács, Markus N. Rabe, and Helmut Seidl. 2012. Model Checking Information Flow in Reactive Systems. In *Verification, Model Checking, and Abstract Interpretation*, Viktor Kuncak and Andrey Rybalchenko (Eds.). Vol. 7148. Springer Berlin Heidelberg, Berlin, Heidelberg, 169–185. https://doi.org/10.1007/978-3-642-27940-9_12

[6] Jeremy Dubreil, Philippe Darondeau, and Herve Marchand. 2010. Supervisory Control for Opacity. *IEEE Trans. Automat. Control* 55, 5 (May 2010), 1089–1100. https://doi.org/10.1109/TAC.2010.2042008

[7] Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12.

[8] Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. 2020. Realizing Omega-Regular Hyperproperties. In *Computer Aided Verification (Lecture Notes in Computer Science)*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer International Publishing, Cham, 40–63. https://doi.org/10.1007/978-3-030-53291-8_4

[9] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. 2018. Synthesizing Reactive Systems from Hyperproperties. In *Computer Aided Verification*, Hana Chockler and Georg Weissenbacher (Eds.). Vol. 10981. Springer International Publishing, Cham, 289–306. https://doi.org/10.1007/978-3-319-96145-3_16

[10] Bernd Finkbeiner and Sven Schewe. 2005. Uniform Distributed Synthesis. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*. IEEE, Chicago, IL, USA, 321–330. https://doi.org/10.1109/LICS.2005.53

[11] Marieke Huisman, Pratik Worah, and Kim Sunesen. 2006. A Temporal Logic Characterisation of Observational Determinism. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*. 13 pp.–3. https://doi.org/10.1109/CSFW.2006.6

[12] Romain Jacob, Jean-Jacques Lesage, and Jean-Marc Faure. 2016. Overview of Discrete Event Systems Opacity: Models, Validation, and Quantification. *Annual Reviews in Control* 41 (Jan. 2016), 135–146. https://doi.org/10.1016/j.arcontrol.2016.04.015

[13] Yiding Ji, Yi-Chin Wu, and Stéphane Lafortune. 2018. Enforcement of Opacity by Public and Private Insertion Functions. *Automatica* 93 (July 2018), 369–378. https://doi.org/10.1016/j.automatica.2018.03.041

[14] Yiding Ji, Xiang Yin, and Stéphane Lafortune. 2019. Opacity Enforcement Using Nondeterministic Publicly-Known Edit Functions. *IEEE Trans. Automat. Control* (2019), 1–1. https://doi.org/10.1109/TAC.2019.2897553

[15] Orna Kupferman and Moshe Vardi. 2001. Synthesizing Distributed Systems. *Proceedings - Symposium on Logic in Computer Science* (April 2001).

[16] Orna Kupferman and Moshe Y. Vardi. 2000. Mu-Calculus Synthesis. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS '00)*. Springer-Verlag, Berlin, Heidelberg, 497–507.

[17] Józef Lubacz, Wojciech Mazurczyk, and Krzysztof Szczypiorski. 2014. Principles and Overview of Network Steganography. *IEEE Communications Magazine* 52, 5 (May 2014), 225–229. https://doi.org/10.1109/MCOM.2014.6815916

[18] Amir Pneuli and Roni Rosner. 1990. Distributed Reactive Systems Are Hard to Synthesize. In *FOCS*. 746–757 vol.2. https://doi.org/10.1109/FSCS.1990.89597

[19] Elissa M. Redmiles. 2020. User Concerns & Tradeoffs in Technology-Facilitated COVID-19 Response. *Digital Government: Research and Practice* 2, 1 (Nov. 2020), 6:1–6:12. https://doi.org/10.1145/3428093

[20] Emmanuel Roche and Yves Schabes (Eds.). 1997. *Finite-State Language Processing*. MIT Press, Cambridge, Mass.

[21] Anooshiravan Saboori and Christoforos N. Hadjicostis. 2007. Notions of Security and Opacity in Discrete Event Systems. In *2007 46th IEEE Conference on Decision and Control*. 5056–5061. https://doi.org/10.1109/CDC.2007.4434515

[22] Anne-Kathrin Schmuck, Thomas Moor, and Rupak Majumdar. 2020. On the Relation between Reactive Synthesis and Supervisory Control of Non-Terminating Processes. *Discrete Event Dynamic Systems* 30, 1 (March 2020), 81–124. https://doi.org/10.1007/s10626-019-00299-5

[23] Yin Tong, Zhiwu Li, Carla Seatzu, and Alessandro Giua. 2018. Current-State Opacity Enforcement in Discrete Event Systems under Incomparable Observations. *Discrete Event Dynamic Systems* 28, 2 (June 2018), 161–182. https://doi.org/10.1007/s10626-017-0264-7

[24] Andrew Wintenberg, Matthew Blischke, Stéphane Lafortune, and Necmiye Ozay. 2021. Enforcement of K-Step Opacity with Edit Functions. In *2021 60th IEEE Conference on Decision and Control (CDC)*. 331–338. https://doi.org/10.1109/CDC45484.2021.9682936

[25] Yi-Chin Wu and Stéphane Lafortune. 2014. Synthesis of Insertion Functions for Enforcement of Opacity Security Properties. *Automatica* 50, 5 (May 2014), 1336–1348. https://doi.org/10.1016/j.automatica.2014.02.038

[26] Yi-Chin Wu, Karthik Abinav Sankararaman, and Stéphane Lafortune. 2014. Ensuring Privacy in Location-Based Services: An Approach Based on Opacity Enforcement. *IFAC Proceedings Volumes* 47, 2 (2014), 33–38. https://doi.org/10.3182/20140514-3-FR-4046.00008

[27] Xiang Yin and Majid Zamani. 2019. Towards Approximate Opacity of Cyber-Physical System: WIP Abstract. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS '19)*. Association for Computing Machinery, New York, NY, USA, 310–311. https://doi.org/10.1145/3302509.3313316

[28] Xiang Yin, Majid Zamani, and Siyuan Liu. 2021. On Approximate Opacity of Cyber-Physical Systems. *IEEE Trans. Automat. Control* 66, 4 (April 2021), 1630–1645. https://doi.org/10.1109/TAC.2020.2998733

A UNFOLDING AND FOLDING STRATEGIES

Proof of Theorem 4.2 (Unfolding). For convenience, we make the following definition. Given a folded trace $f = f_0 f_1 \cdots \in F^\omega$ and its unfolding $t = t_0 t_1 \cdots = \mathbf{unfold}(f)$, we say step n in f corresponds to step j in t if $|\mathbf{unfold}(f_0 \cdots f_{n-1})| < j \leq |\mathbf{unfold}(f_0 \cdots f_n)|$.

Consider an obfuscation policy $\mathbf{obf} : \left(2^{I_{\text{obf}}}\right)^+ \to \left(2^{O_{\text{obf}}}\right)^+$ and inference policy $\mathbf{inf} : \left(2^{I_{\text{inf}}}\right)^+ \to 2^{O_{\text{inf}}}$. Given two folded traces $f, f' \in Tr(\mathbf{obf}, \mathbf{inf})$ let $t = \mathbf{unfold}(f)$ and $t' = \mathbf{unfold}(f')$. Suppose that at some point, the outputs in O_0 differ between t and t' so $j = \min\{k \in \mathbb{N} \mid t_k|_{O_0} \neq t'_k|_{O_0}\}$ exists. Let n denote the corresponding step in the f and f'. If yield differs in t_j and t'_j, then the length of the outputs in f_n and f'_n differ. Otherwise if the outputs in O_{obf} differ in t_j and t'_j, then the outputs themselves differ in f_n and f'_n. Hence the inputs of I_{obf} must differ in f and f' at some step $n' \leq n$ as they result from the deterministic obfuscation policy **obf**. Let $j' = |\mathbf{unfold}(f_0 \cdots f_{n'-1})| + 1$ be the first step in t corresponding to n' in f. By definition of **unfold**, the inputs of I_0 in $t_{j'}$ and $t'_{j'}$ are given by the inputs in $f_{n'}$ and $f'_{n'}$. So $t_{j'}|_{I_0} \neq t'_{j'}|_{I_0}$. By contrapositive, we have the following result

$$\forall p, p' \in \overline{\mathbf{unfold}(Tr(\mathbf{obf}, \mathbf{inf}))} : \; t|_{I_0} = t'|_{I_0} \implies p|_{O_0} = p'|_{O_0} . \tag{31}$$

So given $i \in \left(2^{I_0}\right)^+$, if

$$\exists p = p_0 \cdots p_n \in \overline{\mathbf{unfold}(Tr(\mathbf{obf}, \mathbf{inf}))} : \; p|_{I_0} = i , \tag{32}$$

we can uniquely define $\psi_0(i) = p_n|_{O_0}$, and otherwise we define $\psi_0(i) = \{\text{yield}\}$. Because **inf** is already synchronous, we simply define $\psi_1 = \mathbf{inf}$. Then by construction we see that

$$\mathbf{unfold}(Tr(\mathbf{obf}, \mathbf{inf})) \subseteq P_U \cap Tr(\psi_0, \psi_1) .$$

Conversely, consider a trace $t \in P_U$ but $t \notin \mathbf{unfold}(Tr(\mathbf{obf}, \mathbf{inf}))$. Then as **unfold** is bijective onto P_U, this means that $t = \mathbf{unfold}(f)$ for some $f \notin Tr(\mathbf{obf}, \mathbf{inf})$. As **obf** is defined for all inputs, there exists a trace $f' \in Tr(\mathbf{obf}, \mathbf{inf})$ with the same inputs in I_{obf} as f. Let $t' = \mathbf{unfold}(f')$ and define $j = \min\{k \mid t_k \neq t'_k\}$ which

must exist as $t \neq t'$. Let n denote the corresponding step in the folded system. If yield $\notin t_{j-1} = t'_{j-1}$ then as $t, t' \in P_U$ it must hold that $t_j|_{I_0} = t'_j|_{I_0} = \emptyset$ by the definition of **unfold**. Otherwise if yield $\in t_{j-1} = t'_{j-1}$ then $t_j|_{I_0}$ and $t'_j|_{I_0}$ must be given by the corresponding inputs in f_n and f'_n, respectively. But by assumption, these inputs are equal so $t_j|_{I_0} = t'_j|_{I_0}$. In either case the inputs in I_0 of t and t' are equal up to step j, but the outputs in $O_0 \cup O_1$ are unequal. As $t' \in Tr(\psi_0, \psi_1)$, this implies $t \notin Tr(\psi_0, \psi_1)$. This implies that $\mathbf{unfold}(Tr(\mathbf{obf}, \mathbf{inf})) \supseteq U^\omega \cap Tr(\psi_0, \psi_1)$.

Finally, by definition, traces in P_U always eventually yield and traces outside of P_U must not satisfy the condition of (32) at some point, after which they must always yield by construction. In either case ψ_0 always eventually yields. □

Proof of Theorem 4.2 (Folding). Let $\psi_0 : \left(2^{I_0}\right)^+ \to 2^{O_0}$ and $\psi_1 : \left(2^{I_1}\right)^+ \to 2^{O_1}$ be strategies such that ψ_0 always eventually yields in the sense of (17). As ψ_0 always eventually yields and is defined for all inputs, for every $e = e_0 e_1 \cdots \in \left(2^{O_E}\right)^\omega$, we can inductively construct a trace $t \in Tr(\psi_0, \psi_1)$ so that

$$t|_{O_E} = e_0 \emptyset^{k_0 - 1} e_1 \emptyset^{k_1 - 1} \cdots,$$

for some $k_j > 0$ so that the partial sums $\sum_{j=0}^{n} k_j$ is the index of the n^{th} occurrence of yield in t. For example $k_0 = \min\{k \mid \text{yield} \in \psi_0(e_0 \emptyset^{k-1})\}$. This trace t is unique as it results from the deterministic strategies ψ_0 and ψ_1. Note that t follows the yield behavior, i.e., $t \in P_U$, so there exists $f = f_0 f_1 \cdots \in F^\omega$ such that $t = \mathbf{unfold}(f)$. Additionally, by the construction of t, we see that the plant outputs in O_E of f are exactly e. Furthermore, similar to the proof of the unfolding case, if f' is constructed for plant outputs e' as detailed above, where e' has a common prefix with e, then f' has a corresponding common prefix with f. Hence, we can define the obfuscation policy $\mathbf{obf}(e_0 \cdots e_n) = f_n|_{O_{\text{obf}}}$ and inference policy $\mathbf{inf} = \psi_1$ so that $\mathbf{unfold}(Tr(\mathbf{obf}, \mathbf{inf})) \subseteq Tr(\psi_0, \psi_1) \cap P_U$.

Also similar to the proof of the unfolding case, given a trace $t \in Tr(\psi_0, \psi_1$ that is not constructed as above, at some point t must violate the yield behavior so that $t \notin P_U$. Hence we also have $\mathbf{unfold}(Tr(\mathbf{obf}, \mathbf{inf})) \supseteq Tr(\psi_0, \psi_1) \cap P_U$. Thus ψ_0 and ψ_1 are an unfolding of **obf** and **inf**. □