Integrating Obfuscation and Control for Privacy

Andrew Wintenberg *Graduate Student Member, IEEE*, Necmiye Ozay *Senior Member, IEEE*, Stéphane Lafortune *Life Fellow, IEEE*

***Abstract*— Networked systems must often balance privacy in avoiding leaking sensitive information, with utility in communicating the information that is needed by components to operate correctly. We consider the problem of enforcing privacy and utility with both *obfuscation* (altering communications to mislead eavesdroppers) and *control* (restricting system behavior to avoid information leakage). We present a formulation of this problem which models components of the networked system as interconnected reactive processes. Tools from distributed reactive synthesis are then used to automatically design obfuscators and controllers which coordinate to enforce requirements. In particular we develop formal specifications capturing privacy using the information flow property of *opacity* and utility ensuring availability of information or imposing constraints on the closed-loop system. This synthesis approach is applicable to a large class of network architectures which we demonstrate on three representative problems over a building access control system.**

***Index Terms*— Opacity, Discrete Event Systems, Reactive Synthesis, Distributed Systems, Privacy and Security**

I. Introduction

As barriers to communication are removed by new technologies, more systems are realizing both the benefits and drawbacks of connectivity. This is especially apparent in cyber-physical systems (CPS) which integrate physical processes with dynamics over a computing infrastructure such as a computer network. Many CPS such as location-based services or smart devices on the Internet of Things handle sensitive data that is subject to strict security requirements. Unfortunately, many of these systems are vulnerable to eavesdropping by malicious actors, potentially making them targets for cyber-attacks.

In order to characterize how information flows and leaks out of dynamic systems, many formal security properties have been proposed in computer security, including observational determinism [1] and non-interference [2] for example. In this paper, we consider the privacy property of *opacity*, which characterizes the inability of a passive eavesdropper to infer "secrets" about a system's behaviors [3]. Opacity has been used extensively to express privacy requirements in computer security and it has been applied in control engineering in the areas of CPS [4] and discrete event systems (DES) [5].

This work was supported in part by NSF CPS Award # 1837680, NSF award ECCS-2144416, and a sponsored research award from Cisco Research.

A. Wintenberg, N. Ozay, S. Lafortune are with The University of Michigan, Ann Arbor, MI 48109 USA (e-mail: [awintenb, stephane, necmiye]@umich.edu).

When a system does not preserve a desired privacy property, it is necessary to resort of an enforcement mechanism. While many mechanisms for privacy enforcement have been proposed, they generally take one of two approaches. The first is to alter the *behavior* of the system by feedback control. For example, supervisory control in DES restricts the behaviors that result in information leakage [6]–[8]. The second mechanism is to alter the *observations* of the system, i.e., what is communicated on the network. This often takes the form of encryption, i.e., encoding information in unintelligible streams of outputs. Alternatively, *obfuscation* encodes information in outputs which mimic the original system but aims to enforce privacy from the viewpoint of an eavesdropper. In DES, this takes the form of *edit functions* that selectively insert and delete events output by the system [9]–[11]. Obfuscation may be advantageous or even required in a variety of settings. The implementation of some systems may impose constraints on communication, requiring outputs to preserve their original format. For example, changing the size and type of data reported by the system's sensors may result in unexpected errors in the system's software. Obfuscation also enables information hiding, communicating private information without eavesdroppers even being aware of privacy enforcement. In this case, as long as the enforcement mechanism remains unknown, attackers will not resort to more destructive methods to obtain information from the system. Altering the system's observations may also be required when it is impractical to restrict some aspects of a system's behavior with control, such as human actions in a physical system.

In altering either the system's behavior or observations, privacy enforcement mechanisms must also maintain the system's *utility*, e.g., a controller must maintain safe behavior or observations must enable accurate monitoring. Utility constraints on obfuscators in prior work such as [10] require that observers can infer some specified information about the system's state; however, it is assumed that all observers, both intended and unintended, have the same capabilities and access to information. This may not be the case when controllers and observers are distributed across a network, or when intended recipients need access to sensitive information while unintended ones do not. Our previous work [12] considers the enforcement of privacy and utility in such a distributed setting, but it is limited to obfuscation within a simple linear network topology (pipeline) like the one depicted in Fig. 1. As we shall see, the addition of control in this paper imposes a new challenge for privacy, as an eavesdropper may observe all information transmitted across the network including both sensor outputs and control commands.

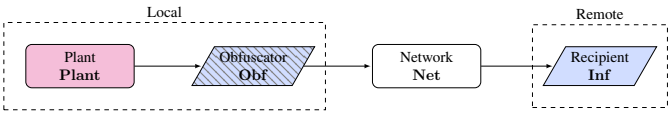


Fig. 1. The architecture for obfuscation and inference without control considered in [12].

In this paper, we address the problem of privacy and utility enforcement with both obfuscation and control in distributed discrete event systems in the context of three general system architectures. We model the dynamic system of interest using finite automata and show how to map privacy and utility enforcement as a as a *distributed reactive synthesis* problem, a problem studied in formal methods and reactive synthesis in computer science [13]. While distributed reactive synthesis is undecidable for general network architectures, we demonstrate our approach on three representative architectures which we show to be decidable. Specifically, we demonstrate how privacy and utility requirements can be expressed with ω-regular specifications over the traces of the automaton model of the system. We express privacy as an extended notion of language-based opacity [14], which has been used to express many other existing notions of opacity in DES [15]–[17]. By explicitly modeling the eavesdropper's beliefs about the system's implementation, our framework can precisely express many practical privacy requirements. Obfuscation is used to enforce privacy and the obfuscator is realized as a finite transducer (or input-output automaton). In parallel, a "decoder," possibly together with a controller, are also synthesized as finite (input-output) automata. The role of the decoder is to provide the additional inference mechanism required by an intended recipient (e.g., a controller) for "de-obfuscation" purposes to address the utility requirement. Thus the obfuscator, the decoder, and the controller (if one needs to be synthesized) constitute a distributed solution, as a set of finite automata that work in tandem for enforcement of privacy and utility.

To summarize, the main contribution of this paper is the formulation of privacy and utility enforcement with obfuscation and control as a distributed reactive synthesis problem and the computation of the solution. This solution takes the form of a set of finite (input-output) automata, which makes it readily implementable. This methodology is demonstrated over three practical network architectures that capture either an existing or to-be-synthesized local or remote controller. To set the stage for the description of the three architectures, consider the following motivating example which will be used as a running example throughout the paper.

Example 1 Consider the building depicted in Fig. 2 whose doors are equipped with keypad sensors and controllable locks. At each door, the keypad reports entry attempts to an authorization server which responds with a signal to open the door or not. These signals may contain sensitive information which raises privacy concerns if the server is remote. For example, an eavesdropper may use their knowledge of the building's layout to deduce room occupancy from their observation of entry attempts at the keypads. While this risk can be mitigated by keeping all information at the local site, it may be infeasible to alter the system's existing network architecture.

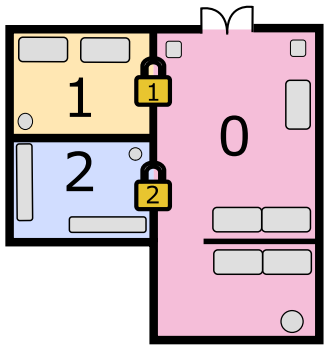


Fig. 2. The layout of a building with two electronically-controlled locked doors. At each door a keypad, shared by both of its sides, controls the lock via a potentially remote authorization server.

In this case, obfuscation can be employed to alter both the keypad outputs sent to the server and control outputs sent back to the building. At the same time, the system must maintain its utility. This may concern a remote user's access to information, for example to diagnose a faulty door. Additionally, utility may require that rooms are accessible by authorized users, i.e., after using the keypad, the door is eventually signaled to open. Our problem is then to design both obfuscators and controllers for a given network architecture to enforce privacy and utility reactively as an individual moves about the building. ♦

The paper is organized as follows. In subsection I-A below, we discuss relevant related works. In Section II, we review concepts from formal languages and discuss our methodology using results from distributed reactive synthesis. Next, we develop our modeling and synthesis approach for privacy and utility enforcement in Section III, over a networked system architecture employing both obfuscation and a local controller (to be synthesized). In Section IV, we consider the second architecture where the controller is no longer local but remote and alongside obfuscation for privacy, the remote controller needs to be synthesized for utility. In the third architecture studied in Section V, the remote controller is now given and it must be secured by a combination of obfuscation and decoding, in order to achieve simultaneously privacy and utility. The solution methodology for synthesizing the automata solutions for the obfuscator, decoder, and controller in all three architectures is explained in Section VI and applied to the building access control example.

A. Related Works

There is a growing literature on privacy enforcement for CPS and DES. In DES, *edit functions* are used to selectively insert or delete events output by the system [9]–[11] in a manner that mimics the original system behavior. In contrast, the notion of event-based cryptography [18] mimics the system's events, but not its dynamics. Alternatively, methods like dynamic masks [19], [20] disable a system's sensors to prevent access to sensitive information by anyone.

As was mentioned earlier, our previous work [12] considers the enforcement of privacy and utility in a pipeline architecture

Control & Obf	CONT	OBS	PRIV	PUB	ASYM	COM
Supervisors [7]	✓		✓	✓		✓
Private Obf [25]		✓	✓			✓
Public Obf [26]		✓	✓	✓		✓
Asym. Obf [12]		✓	✓		✓	✓
Attribute Obf [21]		✓	✓		✓	✓
Co-synthesis [22]	✓[1]	✓	✓		✓	
PPRS [24]	✓		✓		✓	✓
Proposed herein	✓	✓	✓		✓	✓
Other	CONT	OBS	PRIV	PUB	ASYM	COM
Diff. Priv. [27]		✓	✓	✓		✓
Encryption [28]		✓		✓	✓	✓
EBC [18]		✓	✓[2]	✓	✓	✓

[1] Only architecture with control and obfuscation in one feedback loop;
[2] Blocks of outputs are consistent with dynamics, not the entire output string

TABLE I.
TOP PART: A comparison of privacy enforcement works based on obfuscation for CPS and DES.
BOTTOM PART: Other mechanisms for privacy enforcement.
The column labels are as follows: modification of behavior with control CONT, modification of observations OBS, privacy when mechanism is private knowledge PRIV, privacy when mechanism is public knowledge PUB, asymmetric information between recipients ASYM, algorithm is complete COM. Row labels: privacy-preserving reactive synthesis PPRS, event-based cryptography EBC.

without (local/remote) controllers. A similar problem as in [12] with the same topology but with alternative assumptions is also considered in [21].

We summarize and compare in the top part of Table I a set of relevant works on privacy (typically, opacity) enforcement using obfuscation techniques that are related to our paper. It is worth noting that the problem of opacity enforcement considered in [22] utilizes control, obfuscation, and dynamic masks in a fixed network topology. As such, it is closely related to this paper; yet, our contribution differs in a few key aspects. Importantly, our approach is applicable to general network architectures, and furthermore, it is *complete* for the three problems we focus on. In contrast, the synthesis method of [22] is incomplete, potentially not finding a solution when one exists. Secondly, [22] considers control with supervisors which can realize nondeterministic behavior, whereas our controllers are deterministic as implementations of reactive processes. The complex relation between these two control approaches are thoroughly discussed in [23]. Thirdly, [22] employs dynamic masks and obfuscation with general edit functions, while for simplicity we limit our discussion to replacement functions. We also note that the recent work in [24] aims to synthesize an implementation (aka, a controller) that enforces a given (LTL) specification while simultaneously enforcing a second privacy specification; namely, it addresses the problem of privacy-preserving reactive synthesis. However, unlike our set-up, there is no plant model and feedback loop.

The bottom part of Table I lists a few alternative methods for privacy enforcement, for the sake of comparison. A thorough discussion of these methods is beyond the scope of this paper.

II. METHODOLOGY

In this section, we review the topics of formal languages and reactive systems, and then present results on distributed reactive synthesis. A thorough introduction to formal languages can be found in [29] and to discrete event systems in [30]. Results on distributed reactive synthesis are summarized in [13], which are presented here with a similar notation for convenience.

A. Formal Languages

Given a finite alphabet Σ, the set of *finite sequences* and *infinite sequences* are denoted by Σ^* and Σ^ω, respectively. By convention, the set Σ^* contains the *empty sequence* ϵ. The set of *nonempty finite sequences* of Σ is denoted by Σ^+. A *language* is a subset $L \subseteq \Sigma^*$ while an *ω-language* is a subset $M \subseteq \Sigma^\omega$, whose elements are called *strings*. Given a string $t \in \Sigma^*$ with length n, we write $t = t_0, \cdots, t_{n-1}$ where $t_i \in \Sigma$. The set of finite *prefixes* of a string t or infinite string t is denoted by $\overline{t}, \overline{t} \subseteq \Sigma^*$. Likewise, the set of finite prefixes of all strings in a language L or ω-language M is denoted by $\overline{L}, \overline{M} \subseteq \Sigma^*$.

A finite *automaton* is a tuple $G = (Q, \Sigma, \delta, Q_0, Q_m)$ with a finite set of states Q, a finite alphabet Σ, a transition relation $\delta \subseteq Q \times \Sigma \times Q$, initial states $Q_0 \subseteq Q$, and marked or *accepting* states $Q_m \subseteq Q$. A *subautomaton* of G is an automaton whose states and transitions are a subset of those in G. The *size* of an automaton G refers to its number of states $|Q|$. A run of G over a string $t = t_0 t_1 \cdots t_{n-1} \in \Sigma^*$ is a sequence of states $q_0, q_1, \cdots, q_n \in Q$ such that $q_0 \in Q_0$ and for all $j \in \{0, \cdots, n-1\}$, $(q_j, t_j, q_{j+1}) \in \delta$. The run is said to be accepting if it ends in an accepting state, i.e., $q_m \in Q_m$. The *language generated by* G is the set $\mathcal{L}(G) \subseteq \Sigma^*$ containing the strings for which there exists a corresponding run over G. Similarly, the *language accepted by* G is the set $\mathcal{L}_m(G) \subseteq \mathcal{L}(G)$ containing strings with an accepting run. We say a language $L \subseteq \Sigma^*$ is *regular* if it is accepted by some finite automaton. We say that G is *deterministic* if its strings each correspond to a unique run, i.e., if both $|Q_0| \leq 1$ and for all $q \in Q$ and $t \in \Sigma$, there is at most one $q' \in Q$ such that $(q, t, q') \in \delta$.

Similar definitions are made for automata over infinite strings with a Büchi acceptance condition. A *Büchi automaton* is a standard automaton $H = (Q, \Sigma, \delta, Q_0, Q_m)$ with the interpretation that an infinite run is accepting if it visits an accepting state an infinite number of times. Using this, the *ω-language accepted by* H is the set $\mathcal{L}^\omega(H) \subseteq \Sigma^\omega$ of infinite strings with an accepting run over H. We say an ω-language $M \subseteq \Sigma^\omega$ is *ω-regular* if it is accepted by some Büchi automaton. Linear temporal logic (LTL) provides a succinct representation of ω-regular languages using the temporal operators *always* $\Box$, *eventually* $\Diamond$, and *next* $\bigcirc$. A formal presentation of its syntax and semantics can be found in [29]. In a slight abuse of notation we will use the symbol φ to denote a specification given by an ω-regular language or an LTL formula encoding it. When the context is clear, we omit the word *infinite* when referring to infinite strings and refer to ω-languages simply as languages.

B. Distributed Systems

We consider distributed systems composed of interconnected reactive processes. These processes generate outputs

dynamically according to the inputs they have previously received. We model these inputs and outputs by assignments to a subset of Boolean variables V, e.g., for variables $V' \subseteq V$ the set of possible inputs is $2^{V'}$. A behavior of a process with variables $V_p \subseteq V$ evolving over time is described by an infinite sequence or *trace* $t \in \left(2^{V_p}\right)^\omega$. It is often convenient to discuss the restriction of a trace $t = t_0 t_1 \cdots$ to such a subset of variables $V' \subseteq V_p$ which we define as $t|_{V'} = (t_0 \cap V')(t_1 \cap V') \cdots \in \left(2^{V'}\right)^\omega$. We can make similar definitions for the restriction of finite traces and sets of traces. Similarly, it is convenient to discuss the lifting of a set of strings $M \subseteq \left(2^{V_p}\right)^\omega$ to a larger set of variables $V' \supseteq V_p$ which we define by $M|^{V'} = \{t \in \left(2^{V'}\right)^\omega \mid t|_{V_p} \in L\}$. We can make a similar definition for a set of finite traces. A deterministic implementation of a process is a *strategy* $\rho : \left(2^I\right)^+ \to 2^O$ mapping a history of inputs $i_0 \cdots i_n$ to a single output $\rho(i_0 \cdots i_n)$. The set of traces associated with a strategy is defined by

$$\mathrm{Tr}(\rho) = \left\{t \in \left(2^{I \cup O}\right)^\omega \mid \forall n \in \mathbb{N}.\ \rho(t_0 \cdots t_n|_I) = t_n|_O\right\}. \tag{1}$$

Strategies can be represented with automata that accept their set of traces. Such automata representing an input-output relation are often called *transducers*. We say a strategy is *finite* when there exists such an automaton that is finite.

We now present a framework for distributed reactive systems adapted from [13]. There, the system must react to an *unconstrained* environment, producing arbitrary sequences of outputs. As the environment is unconstrained, relations between the environment and the system must be encoded as a kind of *assume-guarantee* specification. As our focus is feedback control of the environment, we present a framework with constrained environments and make this encoding of constraints as specifications explicit. For example, the environment may represent plant under feedback control whose dynamics must be encoded with such a specification.

The arrangement of processes and the interconnection of their inputs and outputs in a distributed system is called an *architecture*. The environment is a special process, with the rest classified as either *white-box* with a fixed, known implementation or *black-box* otherwise. Formally, an *architecture* over a set of variables V is a tuple $A = (P, W, \mathrm{env}, E, O, H)$ with processes P, white-box processes $W \subseteq P$, environment process $\mathrm{env} \in P \backslash W$, and interconnections $E \subseteq P \times P$ forming a directed graph with nodes P and edges E. These edges are labeled by the set of *observable outputs* $O = \{O_e \subseteq V \mid e \in E\}$ communicated along the corresponding connection. Likewise, the nodes are labeled by a set of *hidden outputs* $H = \{H_p \subseteq V \mid p \in P\}$ produced in the corresponding process but not communicated to others. For each process, we require that the observable outputs are disjoint from the hidden outputs. In addition, the set of both observable and hidden outputs should be mutually disjoint for all processes. For convenience we denote the set of outputs, both observable and hidden, for a process $p \in P$ by $O_p = \bigcup_{p' \in P} O_{(p,p')} \cup H_p$. Similarly, we denote the set of inputs of a process $p \in P$ by $I_p = \bigcup_{p' \in P} O_{(p',p)}$. The set of all variables for process p is defined by $V_p = I_p \cup O_p \cup H_p$. *When the context is clear, we may describe the interconnections using only the inputs I_p and outputs O_p of the processes $p \in P$.*

In a given architecture A, white-box processes have a known *implementation* represented as a set of strategies $S_W = \{\rho_p : \left(2^{I_p}\right)^+ \to 2^{O_p} \mid p \in W\}$, whereas non-environment black-box processes will have an unknown implementation denoted by $S = \{\rho_p : \left(2^{I_p}\right)^+ \to 2^{O_p} \mid p \in P \setminus (W \cup \mathrm{env})\}$. We say an implementation is finite if all of its strategies are finite. To model the non-determinism of the environment, we describe its behavior in relation to the other processes as a set of traces $M_{\text{env}} \subseteq \left(2^{V_{\text{env}}}\right)^\omega$. The implementations and environment interact with each other according to the architecture. The behavior of the overall or composed system is described by the set of traces consistent with each process's traces:

$$\mathrm{Tr}(A, S, S_W, M_{\text{env}}) = \bigcap_{\substack{p \in P \\ p \neq \mathrm{env}}} \mathrm{Tr}(\rho_p)|^V \cap M_{\text{env}}|^V. \tag{2}$$

Remark 1 *In this work as in previous work on obfuscation [12], we consider distributed systems without* delay*, i.e., output from one process is available to the next immediately. This is encoded in our definition of traces of a strategy in Equation* (1)*. In general, feedback in systems without delay can result in inconsistent implementations, i.e., a process's input may depend on its outputs in a way that cannot be resolved statically. However, there is a simple transformation from this problem to the one with delay as discussed in [13], [31]. Solutions to the transformed problem represent exactly the consistent solutions to the problem with delay, i.e., those whose behavior can be expressed as a single monolithic implementation.*

C. Synthesis for Distributed Systems

We consider the problem of implementing a distributed system to satisfy a given formal specification. We model these specifications by ω-regular languages $\varphi \subseteq \left(2^V\right)^\omega$ over the set of traces $\mathrm{Tr}(A, S, S_W, M_{\text{env}})$. The problem is then to find an implementation S of the black-box processes so that the composed system traces satisfy the specification.

Problem 1 Given an architecture $A = (P, W, \mathrm{env}, E, O, H)$ over variables V with a white-box implementation S_W and environment traces given by the ω-regular language $M_{\text{env}} \subseteq \left(2^{V_{\text{env}}}\right)^\omega$, find an implementation S for A such that

$$\mathrm{Tr}(A, S, S_W, M_{\text{env}}) \subseteq \varphi. \tag{3}$$

In general, this problem is known to be undecidable [13], [31]; however, there are architectures for which it is decidable. For example, the problem is decidable for *pipeline architectures* which consist of a directed, linear arrangement of processes [31]. More generally it was shown in [13] that the problem is decidable exactly for the architectures without so-called *information forks*. Essentially, an information fork occurs when two processes possess incomparable information, i.e., each process has knowledge about the current behavior

that the other does not. In the absence of such forks, the processes may be ordered by the level of information they possess. Processes with the same information level can simulate each other. This leads to a distributed synthesis algorithm, which allows to show decidability for this class of architectures [13], as summarized in the following result. This result considers environments that are unconstrained, i.e., $M_{\text{env}} = \left(2^{V_{\text{env}}}\right)^{\omega}$, and without input or feedback, i.e., $I_{\text{env}} = \emptyset$.

Theorem 1 (Adapted from Theorem 4.12 [13]) *Let $A = (P, W, env, E, O, H)$ be an architecture over V without information forks such that $M_{env} = \left(2^{V_{env}}\right)^{\omega}$ and $I_{env} = \emptyset$. Let S_W be a finite white-box implementation and $\varphi \subseteq \left(2^{V}\right)^{\omega}$ be an ω-regular language. Then the distributed synthesis problem for A, S_W, M_{env}, φ is decidable.*

Furthermore, if a solution implementation S exists, Section 4 of [13] presents a synthesis algorithm that is guaranteed to find a finite solution. The complexity of the algorithm is related to the number of information levels of the black-box processes. If there are n information levels, the algorithm is n-exponential in the size of the automata representing the specification and implementation of the white-box properties. For example, in a pipeline architecture with 2 black-box processes and a specification represented with an automaton of size m, the complexity would be $O(2^{2^m})$. In what follows, we show how several architectures relevant to privacy-aware control can be transformed to this setting.

Remark 2 *The synthesis algorithm of [13] considers more general specifications given by μ-calculus formulas over the computation tree of the composed system. While in this work we only consider ω-regular specifications for simplicity, our formulation as a distributed synthesis problem may also employ these more general specifications.*

Remark 3 *It is important to note that once the architecture and specifications are formalized in the language of distributed reactive synthesis, as is done in the remainder of the paper, one can use any of the available distributed reactive synthesis methods to obtain correct-by-construction automata that solve the problem at hand [32]–[34]. This is essentially the flexibility of this approach that allows us to solve obfuscation and control problems in a unified way. The details of our specific implementation will be given in Section VI and the Appendix.*

III. Integrating Obfuscation and Control

In this section, we present the first of three problems which integrate obfuscation and control in a networked system to enforce privacy and utility. We will discuss modeling with distributed systems, then specifications for privacy and utility, and finally, a method for synthesis. Our presentation follows a logical progression from the ?local? element of the architecture, where Architecture 1 is the same as in prior work [12] but with the addition of local control, to the ?remote? element of the architecture, as follows: (i) the inclusion of remote control and obfuscation (Architecture 2 in Section IV); and (ii) the securing of an existing remote controller with obfuscation (Architecture 3 in Section V).

First, we build upon the system architecture discussed in [12] and depicted in Fig. 1, in which a plant dynamically produces outputs which are obfuscated before being broadcast on a network and acted upon by a recipient. Here, we additionally consider that some behavior of the plant may be restricted or controlled locally in order to better enforce privacy and maintain utility for the plant and recipient. This controller uses the observable outputs from the plant as feedback to generate inputs to the plant. The components of this networked system including the controller are modeled by processes in a distributed system whose architecture, referred to as Architecture 1, is depicted in Fig. 3. Note that the control action is assumed to be communicated locally, i.e., not broadcast on the network. As such, we are not concerned with the control action leaking information as is considered in Sections IV and V.

A. System Model

The overall system is modeled as a distributed system with architecture $A = (P, W, \text{env}, E, O, H)$. The plant, obfuscator, controller, network, and recipient are represented by the processes $P = \{\mathbf{Plant}, \mathbf{Obf}, \mathbf{Cont}, \mathbf{Net}, \mathbf{Inf}\}$. The interconnection of these processes $E \subseteq P \times P$ is depicted in Fig. 3.

The plant drives the system, nondeterministically producing outputs which must be conveyed to the recipient. As such, it acts as the environment process, i.e., env $= \mathbf{Plant}$. The observable outputs of the plant are communicated to both the obfuscator and controller, i.e., $I_{\text{Obf}} = I_{\text{Cont}} \subseteq O_{\text{Plant}}$, while its hidden outputs represent its internal state. The controller process $\mathbf{Cont}$ provides feedback to the plant $I_{\text{Plant}} = O_{\text{Cont}}$ in order to enforce privacy (e.g., restricting secret-revealing behavior) and utility (e.g., restricting unsafe behavior). We assume the dynamics relating the outputs from the plant to inputs from the controller are described by the ω-regular language M_{Plant} over the plant variables V_{Plant}.

The obfuscator process $\mathbf{Obf}$ modifies the outputs of the plant before they are broadcast on the network to enforce privacy. As the obfuscator seeks to mimic the plant, its outputs are copies of the plant's outputs. Formally, we define $O_{\text{Obf}} = \{o_{\text{Obf}} \mid o \in I_{\text{Obf}}\}$ where o_{Obf} denotes a distinct copy of the plant output variable o which may take on different values. We emphasize that an implementation of the obfuscator is a strategy $\rho_{\text{Obf}} : \left(2^{I_{\text{Obf}}}\right)^{+} \to 2^{O_{\text{Obf}}}$ which in each step replaces a single input from the plant with a single obfuscated output. This corresponds to the notion of a deterministic *edit function* using only replacement [10].

The network broadcasts the outputs it receives from the obfuscator to all recipients on the network, both intended and unintended. In order to capture the potential dynamics of the network, such as a delay or bandwidth limitation, we model the network as a white-box process $\mathbf{Net}$. In Architecture 1 the network receives input from the obfuscator $I_{\text{Net}} = O_{\text{Obf}}$ and transmits copies $O_{\text{Net}} = \{o_{\text{Net}} \mid o \in I_{\text{Net}}\}$. We assume the network has a fixed implementation as a deterministic strategy $\rho_{\text{Net}} : \left(2^{I_{\text{Net}}}\right)^{+} \to 2^{O_{\text{Net}}}$. As such, the network $\mathbf{Net}$ is the only white-box process $W = \{\mathbf{Net}\}$ with $S_W = \{\rho_{\text{Net}}\}$. In the case that the network directly broadcasts its inputs

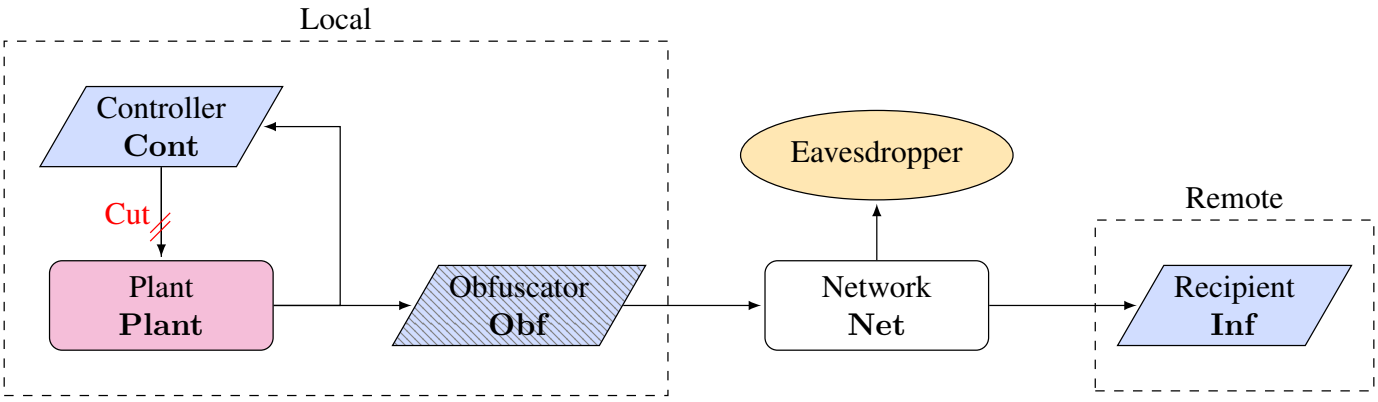


Fig. 3. **Architecture 1** featuring control and obfuscation at the local site which transmit information to the recipient at the remote site. The edge labeled *cut* indicates the feedback eliminated in the transformation used for synthesis in subsection III-D. The black-box processes to be synthesized are represented by parallelograms. The striped parallelograms denote processes unknown by the eavesdropper.

without delay as considered in our examples, we may omit it from the architecture, directly connecting the obfuscator to the recipient.

The final process **Inf** in the system models the actions of the intended recipient of the plant's information, for example *inferring* sensitive plant information. It utilizes the obfuscated outputs from the network $I_{\text{Inf}} = O_{\text{Net}}$ to take action at the remote site modeled by its outputs O_{Inf}.

Example 2 In the building system from Example 1, the authorization server controlling the locks may be local to the building, but outputs from the keypads are shared over a network with their manufacturers in order to diagnose faults requiring maintenance. In particular, door 2 may experience a fault preventing it from opening. We can model this system with Architecture 1 utilizing obfuscation of the keypad signals for privacy. In this model, the plant receives inputs I_{Plant} given by the control $\mathbf{c}^j$ from the server signaling door j to unlock for $j \in \{1, 2\}$. Likewise, the plant produces outputs O_{Plant} consisting of $\mathbf{o}^j$ indicating door j is open, $\mathbf{k}^j$ indicating keypad j is pressed, and $\mathbf{f}$ indicating door 2 is faulty with $j \in \{1, 2\}$.

We assume that locally, the keypad and door outputs $\mathbf{k}^j$, $\mathbf{o}^j$ are observed while the fault $\mathbf{f}$ is hidden. In addition, we assume that the outputs of the plant are delayed by one step before observation which can be represented by introducing delayed copies of these variables; however, in an abuse of notation we simply write $I_{\text{Cont}} = I_{\text{Obf}} = \{\mathbf{k}^j, \mathbf{o}^j\}$. The outputs of the keypads, but not whether the doors are opened, are nominally communicated to the manufacturer to infer a fault with door 2. The obfuscator replicates these outputs over the network with its own set of outputs O_{Obf} given by $\mathbf{k}^j_{\text{Obf}}$ for $j \in \{1, 2\}$. For simplicity, we assume the network communicates these values unmodified without delay, i.e., in an abuse of notation $I_{\text{Net}} = O_{\text{Net}}$. Then finally, the manufacturer is the recipient of these obfuscated outputs $I_{\text{Inf}} = O_{\text{Obf}}$ and produces a single output $O_{\text{Inf}} = \{\mathbf{f}_{\text{Inf}}\}$ whenever the fault has been inferred.

Now we describe the plant dynamics M_{Plant}. We assume the user starting from room 0, always remains at a keypad once pressed until the corresponding door is signaled to open, moving through the door if it has opened. Furthermore, the doors always open immediately once signaled unless the fault forces door 2 to remain closed forever. In order to allow the fault to be diagnosed, we must also assume a kind of liveness that the user always eventually presses accessible keypads, i.e., if the plant visits a state with a transition labeled with $\mathbf{k}^i$ infinitely often, then $\mathbf{k}^i$ must occur infinitely often. The resulting dynamics are represented by the automaton depicted in Fig. 4. ♦

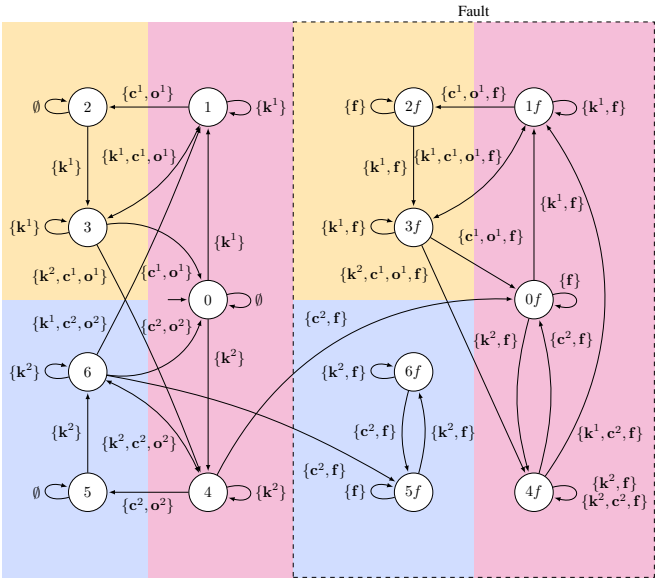


Fig. 4. An automaton encoding the plant M_{Plant} from Example 2. Not depicted are transitions accepting invalid control outputs, e.g., when no keypad is pressed. Background colors indicate which room from Fig. 2 corresponds to each state.

Remark 4 *Beyond replacement, obfuscation can be implemented with edit functions which also delete outputs or insert fictitious ones. In [12], distributed synthesis is used to design obfuscators with insertion. This work transformed the system so the obfuscator outputs one insertion in each step to fit the standard framework of* synchronous *systems considered by distributed synthesis. To this end the obfuscator was augmented with an additional output* **yield** *to indicate*

the end of insertions, and the specification was modified to ensure the plant holds its outputs until **yield** *occurs. This approach is conceptually similar to introducing a local controller as in Architecture 1 with a single output* **yield**. *We may utilize a similar approach to design obfuscators with insertion; however, synchronization is more complex in the presence of multiple feedback paths. So for simplicity, we only discuss obfuscation by replacement.*

B. Privacy Requirements

Knowledge about the current behavior of the plant or system may be used by a malicious agent to damage the system or harm its users. As such, we consider requirements on the privacy of the plant's behavior. We model privacy as the opacity of a set of behaviors identified as *secret*. Opacity requires that the occurrence of these behaviors can never be deduced by an eavesdropper. In particular, we consider the notion of *private safety* [26], where the eavesdropper is aware of the plant's dynamics but not of obfuscation, i.e., the implementation and goals of obfuscation are *private* knowledge. We adapt this notion of private safety to infinite traces over distributed systems with obfuscation and control, expressed as an ω-regular specification. In this more general setting, we assume that the eavesdropper possesses a *nominal model* of the system without obfuscation. Privacy requires that the eavesdropper does not deduce secrets within this model, regardless of what observations are made of the obfuscated system.

In particular, we model these nominal and secret behaviors as ω-languages, adapting of the notion of *language-based opacity* (LBO) [14] to infinite strings. Formally, it is assumed that the eavesdropper's nominal model of the system is given by a known language $\hat{M} \subseteq \left(2^V\right)^\omega$. Within this model, we must ensure they cannot deduce some secret aspects of the behavior given by the *secret language* $M_S \subseteq \left(2^V\right)^\omega$. The strings in the language $\hat{M} \setminus M_S$ are called *nonsecret*. We assume the eavesdropper observes a subset of the output variables $V_{\text{obs}} \subseteq V$ shared between the nominal and actual system, i.e., a string $t \in \left(2^V\right)^\omega$ is observed as $t|_{V_{\text{obs}}}$. Using this nominal model, the eavesdropper deduces that they have observed secret behavior if their observation could not have resulted from nonsecret behavior. Formally, upon the occurrence of the string t, the eavesdropper *cannot* deduce if it was secret if $t|_{V_{\text{obs}}} \in (\hat{M} \setminus M_S)|_{V_{\text{obs}}}$. So we make the following definition for language-based privacy.

Definition 1 (Privacy) Let $\hat{M} \subseteq \left(2^V\right)^\omega$ be the nominal language, $M_S \subseteq \left(2^V\right)^\omega$ the secret language, and $V_{\text{obs}} \subseteq V$ the observed variables. Then we say the language $M \subseteq \left(2^V\right)^\omega$ enforces *privacy* if

$$M|_{V_{\text{obs}}} \subseteq (\hat{M} \setminus M_S)|_{V_{\text{obs}}} . \tag{4}$$

The right side of this inclusion represents observations that are both consistent with the eavesdropper's model of behavior and nonsecret, while the left side of the inclusion represents the true system behavior. From this definition, we can derive the following monotonicity result.

Theorem 2 *Assume that M enforces privacy for secrets M_S in a fixed nominal model $\hat{M}$. If $M' \subseteq M$ and $M'_S \subseteq M_S$ then M' enforces privacy for M'_S.*

Proof: This holds by the properties of set inclusion and difference used in the definition of privacy. ∎

We require that our distributed system enforce privacy, i.e., the infinite traces of the system are consistent with nonsecret behavior.

$$\varphi_{\text{priv}} = \left((\hat{M} \setminus M_S)|_{V_{\text{obs}}}\right)|^V . \tag{5}$$

While it is not the focus of this work, the standard notion of LBO with respect to a language of finite nominal behaviors $\hat{L} = \overline{\hat{M}}$ and finite secret behaviors L_S [14] can be expressed as a safety property φ_{priv} with finite prefixes given by

$$\overline{\varphi_{\text{priv}}} = \left((\hat{L} \setminus L_S)|_{V_{\text{obs}}}\right)|^V . \tag{6}$$

As shown in [15]–[17] many existing notions of opacity may be expressed as LBO. Examples include current-state opacity, initial-state opacity, and notions of K-step opacity.

In general, the nominal model can be any language representing the eavesdropper's beliefs about the system's behavior. These beliefs may be uncertain. For example, an eavesdropper may know the plant behaviors M_{Plant} resulting in a nominal model satisfying $\hat{M}|_{V_{\text{Plant}}} \subseteq M_{\text{Plant}}$. On the other hand, beliefs about the system may also be incorrect. For example, the eavesdropper may be unaware of some processes in the system's architecture, such as the obfuscator. We can model this nominal belief by "shorting out" these processes, treating them as white-box processes with a fixed implementation that directly passes inputs to their corresponding output copies. In this way, we can construct a nominal model over the same variables as the true system model which captures knowledge of the plant dynamics but only a subset of the system's processes denoted $\hat{P}$ as follows.

Definition 2 Given an eavesdropper aware of some nominal processes $\hat{P}$ in the architecture A with plant dynamics M_{Plant}, this knowledge is captured by the *base nominal model* defined by

$$\hat{M}_0 = M_{\text{Plant}}|^V \cap \bigcap_{p \in W \cup \hat{P}} \text{Tr}(\rho_p)|^V , \tag{7}$$

where the implementations ρ_p of processes $p \in P \setminus \hat{P}$ pass their inputs to corresponding output copies.

We demonstrate constructing the privacy specification with the following example.

Example 3 Returning to the building system, we suppose the eavesdropper is unaware of obfuscation in Architecture 1, but is aware of the plant dynamics M_{Plant}. So they model the system as in Fig. 3 with the indicated process **Obf** unknown. Formally, this means $P \setminus \hat{P} = \{\mathbf{Obf}\}$ which defines the base nominal language $\hat{M}_0$ as in Definition 2. Likewise, they may assume that the doors are eventually controlled to open after the keypad is pressed. This requirement is expressed by the LTL formula

$$\varphi_{\text{Cont}} = \bigwedge_{j \in \{1,2\}} \square \left(\mathbf{k}^j \Rightarrow \Diamond \mathbf{c}^j\right) \tag{8}$$

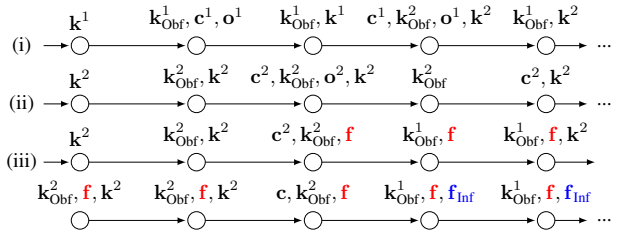


Fig. 5. Possible traces of the building system in Architecture 1. In the first trace, the user passes through door 1 to room 1 and then returns the same way to room 0. During this movement, the obfuscator violates privacy. In step 4, the eavesdropper believes keypad 1 was used immediately after keypad 2 was used from room 0. This could only happen in the original system if a fault prevented door 2 from opening. The remaining traces are drawn from this system implemented as in Fig. 6. The second trace represents non-faulty behavior as the user passes through door 2, whereas the third trace, displayed over two lines, contains the fault $\mathbf{f}$. After the user tries to use door 2 again, the recipient has observed 5 occurrences of $\mathbf{k}^{\mathbf{2}}_{\text{Obf}}$ (an odd number) followed by $\mathbf{k}^{\mathbf{1}}_{\text{Obf}}$, and thus correctly infers a fault has occurred and output $\mathbf{f}_{\text{Inf}}$.

Then the nominal language reflecting their beliefs is given by $\hat{M} = \hat{M}_0 \cap \varphi_{\text{Cont}}$. If the occurrence of the fault $\mathbf{f}$ should be hidden from the eavesdropper for security reasons, we consider the secret language M_{S} expressed by the LTL formula $\Diamond \mathbf{f}$. One can show that this induces the admissible language

$$(\hat{M} \setminus M_{\text{S}})|_{V_{\text{obs}}} = \left(L_1^+ L_2^+\right)^\omega \cup \left(L_2^+ L_1^+\right)^\omega \,, \tag{9}$$

where $L_i = \left(\{\emptyset\}^* \, \{\{\mathbf{k}^i_{\text{Obf}}\}\}^+\right)^2$ corresponds to observations of the user going through door $i \in \{1,2\}$ twice. Roughly this requires the eavesdropper to observe at least two presses of the keypad at a door before one at the other door, and that both keypads are pressed infinitely often. Otherwise, the eavesdropper knows the user left the keypad before the door was opened or that the user was not able to open the door, so a fault is believed to have occurred. This is used to construct the specification φ_{priv} as in Equation (5), which describes traces of the system enforcing privacy. Fig. 5 depicts selected traces of the system when the plant in Fig. 4 is connected to an obfuscator that can successfully enforce privacy and to one that fails to preserve privacy. ♦

Remark 5 *While this framework can include knowledge about obfuscation, conceptually, one must be careful that the nominal model accurately reflects this knowledge. An issue arises if the eavesdropper knows* why *obfuscation is being implemented, i.e., the privacy specification. In this case, a direct description of the nominal model and privacy specification are self-referential: the nominal model encompasses systems satisfying the privacy requirement which is in turn defined with respect to the nominal model. A similar challenge arises when the eavesdropper knows the specification for the controller, which we address in Section IV. One way to resolve the issue with the privacy requirement is to make stronger assumptions on the eavesdropper's knowledge, namely that the implementation of obfuscation is* public knowledge. *Privacy in this case is referred to as* public safety *in [26] which also presents corresponding synthesis methods.*

C. Utility Requirements

Utility refers both to desirable behavior of the plant as well as the recipient's access to information. For example the *utility constraints* proposed in [10], roughly require that all observers unaware of obfuscation are still able to infer which region of states the system currently inhabits. In order to allow for information about the plant to be hidden from unintended observers unaware of obfuscation yet revealed to intended ones that are aware, [12] proposed an alternative utility requirement which we discuss now.

To model this type of specification, we identify a subset of the plant outputs $\mathbf{Data} \subseteq O_{\text{Plant}}$ that should be inferred by the recipient. We model the inference of the recipient explicitly with the output of the process $\mathbf{Inf}$. As such the recipient outputs should match the plant outputs $\mathbf{Data}$. To ensure the output sets are disjoint, we define the outputs $O_{\text{Inf}} = \{o_{\text{Inf}} \mid o \in \mathbf{Data}\}$ as copies of the variables in $\mathbf{Data}$. The utility requirement that the recipient infers the outputs $\mathbf{Data}$ from the plant can be modeled with the LTL formula

$$\varphi_{\text{Data}} = \Box \bigwedge_{o \in \text{Data}} (o \;\Leftrightarrow\; \bigcirc o_{\text{Inf}}) \,. \tag{10}$$

In general, the complete utility specification φ_{util} will be the conjunction of an inference specification φ_{Data} with a control specification φ_{Cont}.

More generally, instead of requiring the recipient to infer the current plant output after a one step delay, we may consider more temporally complex relations as demonstrated in the following example.

Example 4 In the building system, we require that the remote recipient must be able to monitor the building and diagnose door lock faults. Formally, they must eventually infer if $\mathbf{f}$ occurs in the current trace, i.e., $\mathbf{Data} = \{\mathbf{f}\}$. The diagnosis specification can then be expressed with the LTL formula

$$\varphi_{\text{diag}} = \Diamond \mathbf{f} \;\Leftrightarrow\; \Diamond \mathbf{f}_{\text{Inf}} \,. \tag{11}$$

In addition, we will also require that the controller eventually signals the doors to open if the keypads have been pressed. This is captured by the previous specification φ_{Cont} defined in Equation (8). The utility requirement is then the combination

$$\varphi_{\text{util}} = \varphi_{\text{diag}} \cap \varphi_{\text{Cont}} \,. \tag{12}$$

A sample trace of the system when the plant in Fig. 4 is connected to an obfuscator and an inference function that achieves this specification is depicted in Fig. 5 (iii). ♦

D. Synthesis

With this system model and these specifications we can state the design problem for Architecture 1.

Problem 2 (Architecture 1) Given an instance A of Architecture 1 and ω-regular privacy and utility specifications φ_{priv} and φ_{util}, find an implementation S for $\mathbf{Obf}$, $\mathbf{Cont}$, and $\mathbf{Inf}$ solving the distributed synthesis problem for A with specification

$$\varphi = \varphi_{\text{priv}} \cap \varphi_{\text{util}} \,. \tag{13}$$

To apply the distributed synthesis algorithm we must transform our problem to match the conditions of Theorem 1. Specifically, we must eliminate the constraints on the plant and the feedback from the controller while maintaining the same set of solutions. To do this, we utilize two ideas from [13] for simplifying system architectures. First, due to the strict order of informed processes, feedback edges from less informed to more informed processes may be eliminated as they are redundant. Intuitively, as the non-environment processes are deterministic, such feedback outputs from them can simply be predicted. In particular as the environment (the plant in our case) is the most informed processes as the source of nondeterminism in the system, feedback from our controller is redundant. Second, white-box processes may be eliminated by combination with more informed processes and encoding their implementation as part of the specification. Likewise, we can incorporate our plant dynamics in the specification. With these ideas, we transform the architecture and specification to the form used in Theorem 1 without altering the set of solution implementations. We describe this transformation in general now.

Let $A = (P, W, \text{env}, E, O, H)$ be an architecture over V with a finite white-box implementation S_W, environment traces $M_{\text{env}} \subseteq \left(2^{V_{\text{env}}}\right)^\omega$ and an ω-regular specification $\varphi \subseteq \left(2^V\right)^\omega$. We create a new architecture by cutting any feedback to the environment process. Outputs communicated from a process only to the environment are replaced by hidden outputs in the process. For a given process $p \neq \text{env}$, let $H'_p = H_p \cup \left(O_{(p,\text{env})} \setminus \bigcup_{p' \neq \text{env}} O_{(p,p')}\right)$, $O'_{(p,\text{env})} = \emptyset$, and $O'_{(p,p')} = O_{(p,p')}$ for $p' \neq \text{env}$. Likewise, let $E' = \{(p, p') \in E \mid p' \neq \text{env}\}$, $O' = \{O'_e \mid e \in E'\}$, and $H' = \{H'_p \mid p \in P\}$. The transformed architecture is denoted by $A' = (P, W, \text{env}, E', O', H')$. Note that the output set of each process is unchanged, while only the inputs of the environment process were changed by removal. As such, the two architectures support the same implementations.

To capture the environment dynamics M_{env}, observe that the solution only needs to enforce the specification φ over traces agreeing with M_{env}. So we define a new specification

$$\varphi' = \varphi \cup \left(\left(2^V\right)^\omega \setminus {M_{\text{env}}}^{|V}\right) . \tag{14}$$

Thus, this new specification is also ω-regular as it is constructed from the union, complement, and restriction of ω-regular languages. Then as desired, the environment of the transformed system is unconstrained, i.e., ${M_{\text{env}}}' = \left(2^{O_{\text{env}}}\right)^\omega$. We then have the following result.

Theorem 3 *Given a distributed synthesis problem over A, S_W, M_{env}, φ, consider the transformed problem $A', S_W, {M_{env}}', \varphi'$ be constructed as described above. Then the two problems have the same set of solution implementations.*

Proof: Consider an implementation S for both systems. Note that the trace sets for each strategy in the implementations are the same for both architectures. So from the definition of the trace set for distributed systems in equation (2), we see

$$\text{Tr}(A, S, S_W, M_{\text{env}}) = \text{Tr}(A', S, S_W, {M_{\text{env}}}') \cap {M_{\text{env}}}^{|V} .$$

Hence we can compare satisfaction of the specifications:

$$\text{Tr}(A, S, S_W, M_{\text{env}}) \subseteq \varphi \quad \Leftrightarrow$$
$$\text{Tr}(A', S, S_W, {M_{\text{env}}}') \subseteq \varphi \cup \left(\left(2^V\right)^\omega \setminus {M_{\text{env}}}^{|V}\right) = \varphi' .$$

So by the definition of Problem 1, S is a solution for the original problem if and only if it is a solution for the transformed one. ■

As a consequence of this result, if the transformation results in a decidable problem, then we can apply distributed synthesis to find a solution implementation.

Theorem 4 *Problem 2 can be solved in 2-exponential time.*

Proof: After eliminating the feedback in the transformation of Architecture 1, the only potential fork is between the **Obf** and **Cont**. However, as they observe the same inputs and hence possess the same information, this is not an information fork. Thus we may apply the results of Theorem 1 to the transformed architecture. Note the number of the information levels excluding the environment is 2, one for **Inf** and one for both **Obf** and **Cont** as they share the same information. Thus the synthesis algorithm runs in 2-exponential time in the size of the automata representing M_{Plant}, ρ_{Net}, φ_{priv}, and φ_{util}. If a solution implementation S is found, Theorem 3 states that it is also a solution for Problem 2. Likewise if no solution is found, Problem 2 has no solution. ■

Thus distributed synthesis provides a sound and complete method for designing the obfuscator, controller, and actions for the recipient simultaneously to enforce privacy and utility. We detail the specific algorithms we leverage for distributed synthesis in Section VI. We next demonstrate the outcome of the synthesis procedure with the building system.

Example 5 A solution to the distributed synthesis problem for the building system in Architecture 1 is depicted in Fig. 6. In particular, we see that after the corresponding keypad is pressed, the controller immediately opens door 1 while it delays opening door 2. This delay demonstrates coordination of the controller with the obfuscator, allowing the obfuscator to fabricate an extra press of the keypad providing deniability to which room the user is in. Indeed, distributed synthesis could be employed to verify that there exists no solution utilizing control or obfuscation alone. From the solution, we also see that the obfuscator always outputs an even number of presses to keypad 2, unless a fault has occurred, which is precisely what allows the intended recipient to eventually diagnose the fault. These behaviors are demonstrated by the selected traces of the system in Fig. 5. ♦

IV. Remote Network Control and Obfuscation

In some cases, the infrastructure to implement a controller for the plant may only be present at a remote site. As the inputs to the controller and the feedback from the controller are transmitted over the network, they may potentially need to be obfuscated to preserve privacy. In Architecture 2 depicted in Fig. 7, the controller takes the place of the recipient from Architecture 1, implicitly inferring information from obfuscated plant outputs and selecting a control action which is then

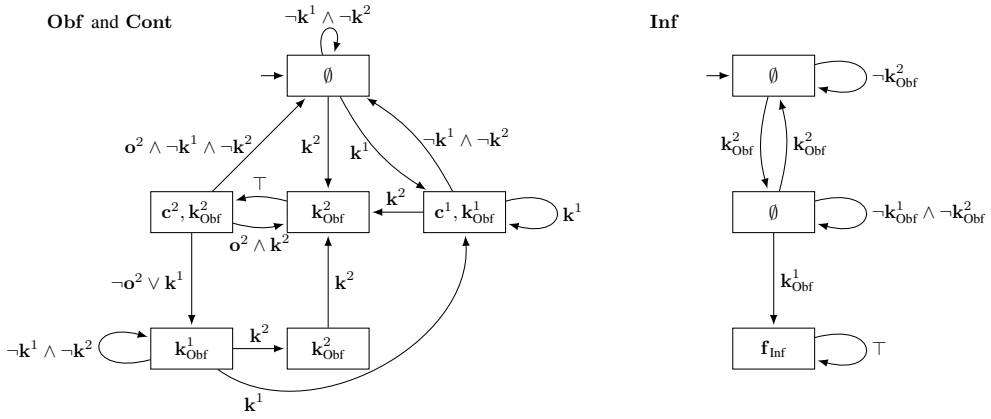


Fig. 6. Automata implementing the combined obfuscator $\mathbf{Obf}$ and controller $\mathbf{Cont}$ (left) and inference function $\mathbf{Inf}$ (right) in the solution to Example 5. For compactness, transitions are labeled by formulas over the plant variables rather than the corresponding sets which satisfy the formulas. The symbol $\top$ denotes *true*, the formula accepting all labels.

obfuscated. To allow the plant to interpret obfuscated outputs from the controller, an additional process called a *decoder* is introduced that processes outputs from the controller before passing them along to the plant.

A. System Model and Specifications

As with Architecture 1, we model the system with a distributed architecture $A = (P, W, \text{env}, E, O, H)$ with processes $P = \{\mathbf{Plant}, \mathbf{Obf}, \mathbf{Net}, \mathbf{Net}_b, \mathbf{Cont}, \mathbf{Dec}\}$ representing the plant, obfuscator, both directions of the network, the controller, and the decoder. Later on, we will discuss why the two channels of the network are modeled with separate processes. The interconnections E are depicted in Fig. 7.

The sets of input and output variables of $\mathbf{Plant}$, $\mathbf{Obf}$, and $\mathbf{Net}$ are the same as in Architecture 1. Now instead of receiving inputs from the plant directly, the controller receives obfuscated inputs from the plant over the network, i.e., $I_{\text{Cont}} = O_{\text{Net}}$. Conversely, it produces outputs which are fed back through the network, i.e. $I_{\text{Net}_b} = O_{\text{Cont}}$. The eavesdropper may observe these control outputs on the network and use them to refine their beliefs about the plant behavior. As such, it may be advantageous for the controller to output obfuscated commands which are then interpreted at the local site by the decoder. So as the control outputs should mimic the plant inputs, we define $O_{\text{Cont}} = \{i_{\text{Cont}} \mid i \in I_{\text{Plant}}\}$. Likewise, the return network process reports the controller outputs to the decoder, so we define $O_{\text{Net}_b} = \{i_{\text{Net}_b} \mid i \in I_{\text{Plant}}\}$. Finally, the decoder de-obfuscates these outputs to provide to the plant with $O_{\text{Dec}} = I_{\text{Plant}}$. Again, the only white-box processes are from the network $W = \{\mathbf{Net}, \mathbf{Net}_b\}$ which each have fixed implementations.

We now discuss the specifics of formulating privacy and utility specifications in this architecture. In particular, we suppose the controller is designed such that the closed-loop behavior of the system satisfies some utility requirement φ_{util} reflecting safety or liveness in the plant for example. As the controller is more apparent in this architecture, the eavesdropper may possess knowledge about both it and the plant. In particular, we assume the eavesdropper knows the utility requirement φ_{util}. We would like to construct the privacy specification φ_{priv} capturing privacy with respect to this new knowledge. Unaware of obfuscation, the eavesdropper derives their nominal model from the plant dynamics and Architecture 2 in Fig. 7 with the indicated unknown processes $P \setminus \hat{P} = \{\mathbf{Obf}, \mathbf{Dec}\}$. This defines the base nominal model $\hat{M}_0$ as in Definition 2. At this point, the eavesdropper expects behavior from $\hat{M}_0$ that not only satisfies φ_{util}, but belongs to an implementation solving the corresponding distributed synthesis problem for Architecture 2 over the nominal processes $\hat{P}$. The correct nominal model should then consist of the union of the trace sets of all such solutions.

In this case where the plant is in a simple feedback loop with the controller, we can use ideas from ω-supervisory control [35] to construct the nominal model $\hat{M}$ as an ω-regular language. If the plant is completely observed by the controller, the union of all deterministic solutions to the reactive synthesis problem form a maximally permissive supervisor. Specifically, we take $\hat{M}$ to be the supremal closed-loop behavior with respect to $\hat{M}_0$ describing the *plant* and the language φ_{util} describing the specification. More generally in the partial observation case, there is no unique maximal solution. However, for regular specifications over finite strings, the union of all correct supervisors can be constructed as a regular language as shown in [36] and Theorem 2 of [37]. Under mild conditions, these results may be extended to the ω-regular case. With this, we can now discuss the building example in detail.

Example 6 We now consider that the building is instead controlled remotely as in Architecture 2 subject to the same specification φ_{Cont} from Example 3. Removing the requirement that the remote site infers the fault, the utility specification is simply $\varphi_{\text{util}} = \varphi_{\text{Cont}}$. We construct the privacy specification in a manner similar to before, identifying the fault as the secret behavior and assuming the processes related to obfuscation, i.e., $\mathbf{Obf}$ and $\mathbf{Dec}$, are unknown to the eavesdropper. The key differences are that now the eavesdropper observes the control actions over the network and knows about the control specification φ_{util}. Thus the nominal model is constructed as all closed-loop behaviors of all implementations ensuring the base

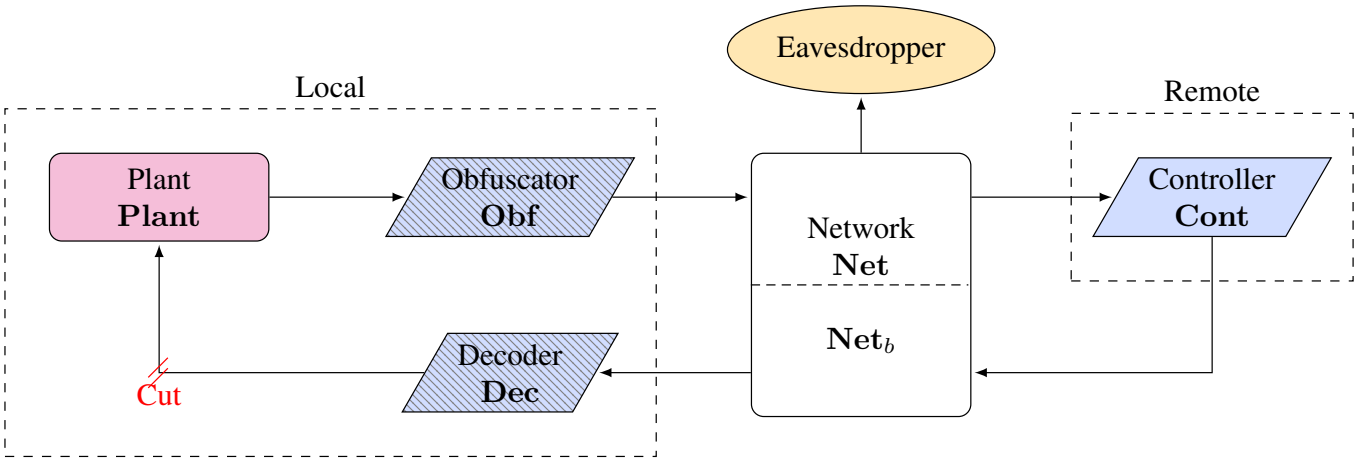


Fig. 7. **Architecture 2** featuring a controller at the remote site which operates on obfuscated data produced by the obfuscator and consumed by the decoder at the local site. The processes are styled as in Fig. 3.

nominal model $\hat{M}_0$ satisfies the specification φ_{util} as discussed above. In this case, it turns out that this nominal model $\hat{M}$ and secret language L_{S} are the same as the one from Example 3 for Architecture 1 (up to the addition and renaming of variables). However, as the obfuscated controls $\mathbf{c}^1_{\text{Obf}}$ and $\mathbf{c}^2_{\text{Obf}}$ are now observed by the eavesdropper, the admissible observations are different, now given by

$$(\hat{M} \setminus M_{\text{S}})|_{V_{\text{obs}}} = \left(\left(L_1^+ L_2^+\right)^\omega \cup \left(L_2^+ L_1^+\right)^\omega\right) , \qquad (15)$$

where $L_i = \left(\{\emptyset\}^* \ \{\{\mathbf{k}^i_{\text{Obf}}\}\}^* \{\{\mathbf{c}^i_{\text{Obf}}, \mathbf{k}^i_{\text{Obf}}\}\}\right)^2$ corresponds to observations of the user going through door $i \in \{1, 2\}$ twice. This defines the privacy specification φ_{priv} as in Equation (5). ♦

B. Synthesis

With the system model and specifications we can state the design problem for Architecture 2.

Problem 3 (Architecture 2) Given an instance A of Architecture 2 and ω-regular privacy and utility specifications φ_{priv} and φ_{util}, find an implementation S for **Obf**, **Cont**, and **Dec** solving the distributed synthesis problem for A with specification $\varphi = \varphi_{\text{priv}} \cap \varphi_{\text{util}}$.

As before, we can transform the system with Theorem 3 to match the conditions of Theorem 1.

Theorem 5 *Problem 3 can be solved in 3-exponential time.*

Proof: After eliminating the feedback from the decoder as depicted in Fig. 7, the resulting architecture is a pipeline, free of information forks. Thus we may apply the results of Theorem 1 to the transformed architecture. As there are 3 black-box processes in the pipeline, the synthesis algorithm runs in 3-exponential time in the size of the automata representing M_{Plant}, ρ_{Net}, ρ_{Net_b}, φ_{priv}, and φ_{util}. ■

Here we see why the two channels of the network must be modeled as separate processes. If instead they are modeled with a single process, there is an information fork between the decoder and controller rooted at the hypothetical merged network process. This is because the network may transmit different information from the plant to the controller and decoder. Alternatively, we can also avoid information forks with a single network process by requiring it to transmit the same outputs to each receiving process.

Example 7 A solution to the distributed synthesis problem for the building system with Architecture 2 is depicted in Fig. 8. From the solution, we see the obfuscator communicates to the controller the use of keypad 1 by the absence of output and of keypad 2 by two consecutive $\mathbf{k}^2_{\text{Obf}}$. Similarly, the controller communicates the open command to the decoder for door 1 by the absence of output and for door 2 by two consecutive $\mathbf{c}^2_{\text{Obf}}$. All the while these processes intersperse the outputs $\mathbf{k}^1_{\text{Obf}}$ and $\mathbf{c}^1_{\text{Obf}}$ to mimic the nominal system without obfuscation. In addition by removing fault diagnosis from the utility specification, we can observe that the remote site can no longer infer the occurrence of the fault. ♦

V. Securing an Existing Remote Controller with Obfuscation

In this section, as in the previous one, we consider a plant that is controlled from the remote site. However, we assume that this controller has an existing implementation which cannot be altered. As this implementation may not have been designed with security in mind, it may leak sensitive information. We now consider the problem of securing such a controller by obfuscating its outputs in addition to those of the plant while maintaining the original closed-loop behavior. As in Architecture 2, the obfuscated controller outputs must be de-obfuscated before they can be input into the plant. Conversely, because the controller has a fixed implementation that was designed without obfuscation, its inputs must also be de-obfuscated. Unlike Architecture 2, for simplicity we model both obfuscation and decoding with a single process at each site. Similarly, we model both channels of the network with a single process as well. By merging these processes, we can avoid the information fork between the obfuscator and decoder present in Architecture 2. While both of these processes perform obfuscation and decoding, for consistency

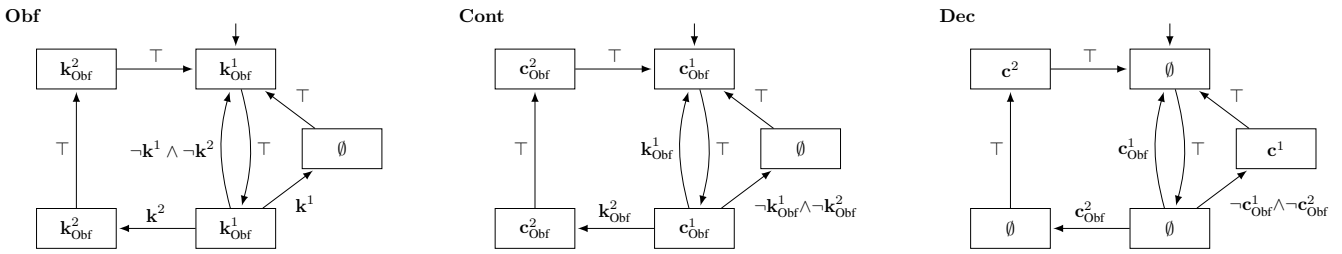


Fig. 8. Automata implementing the obfuscator **Obf** (left), controller **Cont** (middle), and decoder **Dec** (right) in the solution to Example 7.

with the Architecture 2 we will refer to the local process as the obfuscator **Obf** and the remote process as the decoder **Dec**. We refer to this architecture as depicted in Fig. 9 as Architecture 3.

A. System Model and Specifications

We model the system with a distributed architecture $A = (P, W, \text{env}, E, O, H)$ with processes $P = \{\mathbf{Plant}, \mathbf{Obf}, \mathbf{Net}, \mathbf{Dec}, \mathbf{Cont}\}$ representing the plant, obfuscator, network, obfuscator, and controller. These obfuscator processes also take the role of the decoder from Architecture 2. The interconnection of these processes E is depicted in Fig. 9.

While the plant remains unchanged, the obfuscator, decoder, and network now receive inputs and produce outputs mirroring both the inputs and outputs of the plant. Formally, $O_{\text{Obf,Net}}$, $O_{\text{Net,Dec}}$, and $O_{\text{Dec,Cont}}$ are distinct copies of O_{Plant}. Likewise $O_{\text{Net,Obf}}$, $O_{\text{Dec,Net}}$, and $O_{\text{Cont,Dec}}$ are distinct copies of I_{Plant}.

Unlike in the previous architectures, the controller is added as a white-box process $W = \{\mathbf{Cont}, \mathbf{Net}\}$ with a fixed implementation $\rho_{\text{Cont}} : \left(2^{I_{\text{Cont}}}\right)^+ \to 2^{O_{\text{Cont}}}$. We assume that the closed-loop behavior for this controller must be maintained by the obfuscators. We can express this utility requirement with a specification φ_{util} ensuring the plant and controller inputs are exactly recovered by corresponding decoders after obfuscation. We can express this with an LTL formula

$$\varphi_{\text{util}} = \bigwedge_{i \in I_{\text{Obf}}} \underbrace{\square(i \leftrightarrow \bigcirc i_{\text{Cont}})}_{\substack{\text{Control input is} \\ \text{delayed plant output}}} \wedge \bigwedge_{i \in I_{\text{Plant}}} \underbrace{\square(i \leftrightarrow i_{\text{Dec}})}_{\substack{\text{Plant input is} \\ \text{control output}}} . \tag{16}$$

Example 8 We again consider the building utilizing a controller at the remote site; however, we now assume that its implementation is fixed due to practical considerations. This implementation simply immediately sends the signal for the door to open once the corresponding keypad signal is received. Without obfuscation, this controller satisfies the utility specification from Example 6, requiring doors be signaled to open after the keypad is pressed. We will assume that this specification forms the eavesdropper's knowledge about the controller. Given the processes related to obfuscation, i.e., $P \setminus \hat{P} = \{\mathbf{Obf}, \mathbf{Dec}\}$, are unknown to the eavesdropper, this results in the same privacy specification φ_{priv} as in Example 6 (up to renaming variables). In order to maintain the existing closed-loop behavior, we must ensure the plant and controller recover their respective inputs exactly from obfuscation. As explained above, this is captured by the utility specification φ_{util} from Equation (16). ♦

B. Synthesis

With the system model and specifications we can state the design problem for Architecture 3.

Problem 4 (Architecture 3) Given an instance A of Architecture 3 and ω-regular privacy and utility specifications φ_{priv} and φ_{util}, find an implementation S for **Obf** and **Dec** solving the distributed synthesis problem for A with specification $\varphi = \varphi_{\text{priv}} \cap \varphi_{\text{util}}$.

As before, we can transform the system with Theorem 3 to match the conditions of Theorem 1.

Theorem 6 *Problem 4 can be solved in 2-exponential time.*

Proof: While there are many forks in this architecture, none of them constitute information forks. This is because information from the plant, i.e., the environment, propagates linearly to the other processes inducing a strict order on the processes levels of information. Indeed, after removing the feedback edges which are redundant and applying the transformation in Theorem 3, the resulting architecture is a pipeline. Thus we may apply the results of Theorem 1 to the transformed architecture. As there are 2 black-box processes in the pipeline, the synthesis algorithm runs in 2-exponential time in the size of the automata representing M_{Plant}, ρ_{Net}, ρ_{Cont}, φ_{priv}, and φ_{util}. ■

Example 9 Analysis of the building system with Architecture 3, shows that there are in fact no solutions to the distributed synthesis problem. We note that while under non-faulty conditions, there are enough possible messages that the obfuscator can send to convey which key pads have been pressed: the obfuscator can transmit its inputs without modification. However, such a solution is no longer possible once we consider the occurrence of a fault. This possibility in the source of information, i.e., the plant, along with decrease in bandwidth from mimicking non-faulty behavior renders the problem unfeasible. Alternatively, if we assume that the fault has already occurred which reduces the amount of information that needs to be conveyed, a solution exists which is depicted in Fig. 10. In fact this solution guarantees privacy even when the eavesdropper knows the implementation of the controller. ♦

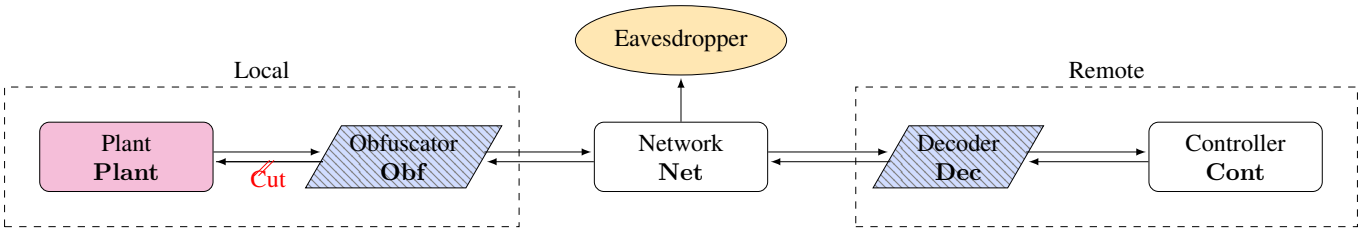


Fig. 9. **Architecture 3** featuring a controller with a fixed implementation at the remote site which must be secured by combination obfuscator-decoders at both the local and remote site. The processes are styled as in Fig. 3.

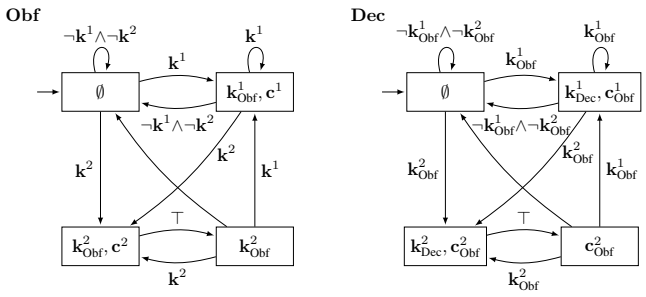


Fig. 10. Automata encoding the implementation of the obfuscator **Obf** (left) and decoder **Dec** (right) for Example 9.

Remark 6 *Instead of combining obfuscation and decoding into a single process in Architecture 3, we may instead maintain two separate processes like in Architecture 2. This alternative architecture is depicted in Fig. 11. While conceptually similar, there are a number of practical differences between these architectures. This alternative ensures, for example, that the local decoder only uses information transmitted from the remote obfuscator. In contrast, decoding at the local site in Architecture 3 may use all information available locally, including the direct outputs from the plant. In addition, the alternative architecture contains 4 black-box process arranged linearly after eliminating feedback to the environment. As a result, the synthesis algorithm for the alternative requires 4-exponential time, compared to the 2-exponential time required for Architecture 3.*

VI. Implementation Details

In this section, we describe how distributed reactive synthesis problems can be solved in practice. In particular, we explain at a high level the approach used to design the solutions for the building access control examples presented in the previous sections. Additional details about the implementation can be found in the appendix.

We observed that the original algorithm for distributed synthesis proposed in [13], [31] does not scale to the example problems considered here. This is due in part to the algorithm's explicit construction of automata of n-exponential size. Instead, we employed a similar approach to [12] used for the synthesis of obfuscators and inference functions. This approach is based upon the reduction from distributed synthesis to synthesis for hyperproperties described in [38]. *Hypeproperties* generalize the concept of specifications for individual traces such as LTL properties, to relations of multiple traces.

Arch.	States	Hyper States	Synth. Time (s)
1	59	2	149
2	81	3	72
3	31	2	548

TABLE II. Information for the synthesis of the reduced examples for each architecture, including the number of states in the automata describing the property and hyperproperty specifications.

In short, the reduction constructs a hyperproperty encoding the information flow induced by the distributed architecture as a relation of the input and output variables of different traces.

We then solved the reduced synthesis problem for hyperproperties using the tool BoSyHyper [39]. This tool achieves improved performance by taking advantage of the existence of small solutions with bounded synthesis as well as advanced heuristics employed within modern constraint solvers. Unfortunately, even with these optimizations, the tool was unable to synthesize solutions for the building access control problems discussed in Examples 5, 7, and 9. In order to obtain solutions for these problems, we manually abstracted the plant and specifications, simplifying the problems and reducing their complexity for the synthesis tool. The tool was then able to synthesize solutions for the reduced problem which were then manually lifted to the original problems. These solutions, depicted in Fig. 6, 8, and 10, were then formally verified for correctness in enforcing privacy and utility. Information about the synthesis for the reduced problems is provided in Table II. Counterintuitively, the smaller sized examples result in longer synthesis times. This demonstrates the observed fact that smaller problem sizes do not necessarily correspond to easier problems for the constraint solvers.

In addition to improved performance, the hyperproperty approach is also more extensible. While the original explicit algorithm cannot be applied to architectures with an information fork, the information flow of arbitrary distributed architectures can be expressed with hyperproperties. This problem is undecidable in general; however, synthesis algorithms for hyperproperties provide a sound but incomplete method for more general architectures. In our framework this would enable synthesis for problems with non-deterministic network delays as well as plants distributed across multiple sites.

VII. Conclusion

In this work, we have addressed the problem of enforcing privacy and utility over networked systems with both obfuscation and control. The joint consideration of obfuscation and control (present in a feedback loop), along with the

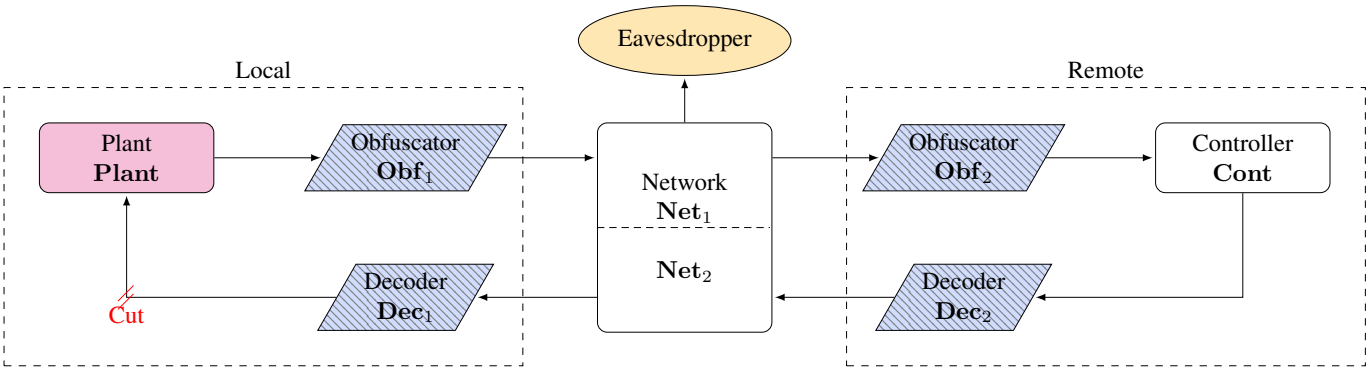


Fig. 11. An alternative to Architecture 3 for securing an existing network controller.

completeness property of our solution procedure, distinguish our results from prior work. Specifically, we considered three distinct networked system architectures with feedback, with a local or a remote controller given or to-be-designed. In each case, we showed how to map our synthesis problem, which involves an obfuscator, a controller, and a decoder, to a decidable instance of a distributed reactive synthesis problem. We were therefore able to leverage the algorithmic solution procedures and tools for finite automata system models in distributed reactive synthesis to solve the three architectures, i.e., to synthesize automata implementations for the obfuscator, controller, and decoder.

There are many directions for future work. The primary limitation for the application of our approach is the scalability of distributed reactive synthesis. This is an active area of research where a variety of techniques are being investigated to design more efficient algorithms. For example, recently proposed algorithms take advantage of a system's modularity [40] or analyze the information flow needed to satisfy the system's specifications [41]. While not considered here, stochastic systems and stochastic privacy requirements are frequently encountered in practice. For example while an eavesdropper may not be able to disprove behavior was not secret, they may use stochastic knowledge about the system to infer the corresponding probability is vanishingly small. It may be possible to adapt our framework to this setting using stochastic notions of opacity for privacy like those proposed in [42], [43].

In addition, while our approach enforces privacy under the assumption that the eavesdropper is unaware of the goals of obfuscation, stronger security guarantees may be required in practice. In particular, we may assume that the attacker knows everything about the system and specifications except for its specific implementation just as encryption algorithms typically assume the attacker knows everything except the key. Future work should account for privacy against such obfuscation-aware eavesdroppers.

APPENDIX

This appendix details the synthesis of solutions to the building access control problems presented in Examples 5, 7, and 9.

A. Specifications for Examples

We now discuss the trace specifications for the example problems in detail. In particular, we represent the specifications with Büchi automata as needed by the synthesis tool. The desired specification used for each of the three examples is of the form $\varphi = \varphi_{\text{util}} \cap \varphi_{\text{priv}}$ as in Equation (13). Here φ_{util} describes utility requirements for each problem while privacy is described by $\varphi_{\text{priv}} = \left((\hat{M} \setminus M_{\text{S}})|_{V_{\text{obs}}}\right)|^{V}$ as in Equation (5) depending on the nominal model $\hat{M}$. In light of Theorem 3, the specification input to the synthesis tool must incorporate the plant dynamics M_{Plant} constructed as in Equation (14) as $\varphi' = \varphi \cup \left(\left(2^{V}\right)^{\omega} \setminus M_{\text{env}}|^{V}\right)$.

Each example utilizes the same plant dynamics M_{Plant} encoded by the automaton depicted in Fig. 4 which represents a single user's movement throughout the building. The liveness condition that all persistently accessible keypads are eventually used can be expressed as a Streett acceptance condition over the states of this automaton. Formally, for $j \in \{1, 2\}$ we define B_j as the set of states where keypad j can be pressed, i.e., states with an outgoing transition labeled by $\mathbf{k}^j$. Specifically, $B_1 = \{0, 1, 2, 3, 0f, 1f, 2f, 3f\}$ and $B_2 = \{0, 4, 5, 6, 0f, 4f, 5f, 6f\}$. Likewise, we define G_j as the set of states where keypad j is pressed, i.e., the destination of these transitions labeled by $\mathbf{k}^j$. Specifically, $G_1 = \{1, 3, 0f, 3f\}$ and $G_2 = \{4, 6, 4f, 6f\}$. The acceptance condition for liveness requires for each j, that if the states of B_j are visited infinitely often then so are the states of G_j. In words, if keypad j can always eventually be pressed, then it is always eventually pressed. It is well-known that such Streett conditions can be transformed into a Büchi conditions [44] which are used by the synthesis tool.

Next, we discuss the construction of the privacy specification. In all three examples, we assume the eavesdropper's model of the system architecture is the plant with dynamics M_{Plant} in direct feedback with the controller with a unit delay. As such, the base nominal models $\hat{M}_0$ in each problem are the same up to the renaming of variables. Furthermore, we assume that the eavesdropper knows that the closed-loop system satisfies the control specification φ_{Cont} defined in Equation (8) which requires door j to eventually be signaled to open with $\mathbf{Cont}^j$ after the corresponding keypad has been pressed with output $\mathbf{k}^j$. In this case, we know that feasible traces must

belong to the intersection $\hat{M} = \hat{M}_0 \cap \varphi_{\text{Cont}}$ which is used by the eavesdropper as their nominal model in the example for Architecture 1.

However, the question remains whether all of such traces of the plant satisfying the control specification are realized by a controller, as pondered by the more astute eavesdroppers considered in Architecture 2 and Architecture 3. In this case, the question can be answered in the affirmative by inspection, any trace in $\hat{M}_0 \cap \varphi_{\text{Cont}}$ can be achieved by a controller by appropriately inserting delays in opening doors as necessary. We see that the eavesdroppers in all three examples utilize the same nominal model which can be expressed by the plant automaton with an additional Streett acceptance condition modeling φ_{Cont}. In particular given the secret language M_{S} defined by the occurrence of the fault, the language $\hat{M} \setminus M_{\text{S}}$ can be expressed by the non-faulty (left) half of the plant automaton where each room must be visited infinitely often. This is described by the Streett condition with B_i is given by all states and G_i is given by the states of room $i \in \{0, 1, 2\}$. Specifically, $G_0 = \{0, 1, 4\}$, $G_2 = \{2, 3\}$, and $G_2 = \{5, 6\}$. Again, this Streett automaton is then converted to a Büchi automaton. Finally, a single Büchi automaton accepting φ' may be constructed using the standard constructions for the complement and intersection.

B. Synthesis of Examples

Next, we present the details of how solutions for the example problems were designed. While the classical tree-automaton based algorithm [13], [31] is useful for theoretic analysis of distributed reactive synthesis, its explicit construction of large automata results limits applicability. Unfortunately, there are not many tools available for the purpose of distributed synthesis. Alternatively, solving the reduction to the more general problem of synthesis for hyperproperties has been observed to improve performance. For example, this is the approach taken in [12] to synthesize obfuscation and inference functions for privacy and utility enforcement. Hyperproperties generalize the concept of trace specifications such as LTL or ω-regular properties. Whereas properties describe individual traces, hyperproperties describe relations of multiple traces. For example, HyperLTL is a formal logic for expressing hyperproperties which extends LTL with explicit trace quantifiers [45]. It is capable of expressing classical information flow properties such as non-interference as well as information flow within a distributed architecture [39].

Using this fact, we can reduce the distributed synthesis problem to a HyperLTL synthesis problem by introducing a HyperLTL formula capturing the architecture. This formula is built using the following HyperLTL formula which expresses the causal dependence of the output variables $O \subseteq V$ on the input variables $I \subseteq V$ without delay

$$\mathrm{D}_{I \to O} = \forall \pi. \forall \pi'. \Big(\bigvee_{o \in O} O[\pi] \leftrightarrow O[\pi'] \Big) \mathcal{W} \Big(\bigvee_{i \in I} I[\pi] \leftrightarrow I[\pi'] \Big),$$

where $\mathcal{W}$ denotes the weak until operator. In words, this formula requires the outputs of any two traces to be the same until their inputs differ. A similar definition for $\mathrm{D}_{I \to O}$ is made in [39] for processes with delay. Given a distributed architecture $A = (P, W, \text{env}, E, O, H)$, we can construct a HyperLTL formula encoding its information flow as

$$\Phi_A = \bigwedge_{\substack{p \in P \\ p \neq \text{env}}} \mathrm{D}_{I_p \to O_p}. \tag{17}$$

We then aim to solve the synthesis problem for the trace property φ' and the hyperproperty Φ_A.

We solve this problem using the tool BoSyHyper [38], an extension of the well-known LTL bounded synthesis tool BoSy to HyperLTL [46]. In particular BoSyHyper supports synthesis for HyperLTL formulas with only universal quantifiers, which includes the formulas needed for distributed synthesis. The tool was modified to accept as input the trace specification φ' represented by an explicit Büchi automaton and the HyperLTL formula Φ_A. If found by the tool, solutions are given as a monolithic model of the realized system in the Aiger format [46]. Such solutions can be converted into automata models implementing each process of the system. The construction and manipulation of automata input and output by BoSyHyper was performed with the automata library M-DESops [47].

REFERENCES

[1] M. Huisman, P. Worah, and K. Sunesen, "A temporal logic characterisation of observational determinism," in *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, Jul. 2006, pp. 13 pp.–3.

[2] S. Zdancewic and A. C. Myers, "Secure Information Flow and CPS," in *Programming Languages and Systems*, ser. Lecture Notes in Computer Science, D. Sands, Ed. Berlin, Heidelberg: Springer, 2001, pp. 46–61.

[3] J. Bryans, M. Koutny, and P. Ryan, "Modelling Opacity Using Petri Nets," *Electr. Notes Theor. Comput. Sci.*, vol. 121, pp. 101–115, Feb. 2005.

[4] X. Yin, M. Zamani, and S. Liu, "On Approximate Opacity of Cyber-Physical Systems," *IEEE Transactions on Automatic Control*, vol. 66, no. 4, pp. 1630–1645, Apr. 2021.

[5] R. Jacob, J.-J. Lesage, and J.-M. Faure, "Overview of discrete event systems opacity: Models, validation, and quantification," *Annual Reviews in Control*, vol. 41, pp. 135–146, Jan. 2016.

[6] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau, "Concurrent Secrets," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 425–446, Dec. 2007.

[7] J. Dubreil, P. Darondeau, and H. Marchand, "Supervisory Control for Opacity," *IEEE Transactions on Automatic Control*, vol. 55, no. 5, pp. 1089–1100, May 2010.

[8] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Current-state opacity enforcement in discrete event systems under incomparable observations," *Discrete Event Dynamic Systems*, vol. 28, no. 2, pp. 161–182, Jun. 2018.

[9] Y.-C. Wu and S. Lafortune, "Synthesis of insertion functions for enforcement of opacity security properties," *Automatica*, vol. 50, no. 5, pp. 1336–1348, May 2014.

[10] Y.-C. Wu, V. Raman, B. C. Rawlings, S. Lafortune, and S. A. Seshia, "Synthesis of Obfuscation Policies to Ensure Privacy and Utility," *Journal of Automated Reasoning*, vol. 60, no. 1, pp. 107–131, Jan. 2018.

[11] Y. Ji, Y.-C. Wu, and S. Lafortune, "Enforcement of opacity by public and private insertion functions," *Automatica*, vol. 93, pp. 369–378, Jul. 2018.

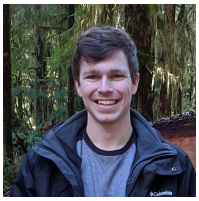
[12] A. Wintenberg, M. Blischke, S. Lafortune, and N. Ozay, "A Dynamic Obfuscation Framework for Security and Utility," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, May 2022, pp. 236–246.

[13] B. Finkbeiner and S. Schewe, "Uniform Distributed Synthesis," in *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*. Chicago, IL, USA: IEEE, 2005, pp. 321–330.

[14] F. Lin, "Opacity of discrete event systems and its applications," *Automatica*, vol. 47, no. 3, pp. 496–503, Mar. 2011.

[15] Y.-C. Wu and S. Lafortune, "Comparative analysis of related notions of opacity in centralized and coordinated architectures," *Discrete Event Dynamic Systems*, vol. 23, no. 3, pp. 307–339, Sep. 2013.

[16] A. Wintenberg, M. Blischke, S. Lafortune, and N. Ozay, "A general language-based framework for specifying and verifying notions of opacity," *Discrete Event Dynamic Systems*, vol. 32, no. 2, pp. 253–289, Jun. 2022.
[17] J. Balun and T. Masopust, "Comparing the notions of opacity for discrete-event systems," *Discrete Event Dynamic Systems*, vol. 31, no. 4, pp. 553–582, Dec. 2021.
[18] P. M. Lima, L. K. Carvalho, and M. V. Moreira, "Networked Automation Systems: A new cryptographic scheme," *Simpósio Brasileiro de Automação Inteligente - SBAI*, vol. 1, no. 1, Oct. 2021.
[19] F. Cassez, J. Dubreil, and H. Marchand, "Synthesis of opaque systems with static and dynamic masks," *Formal Methods in System Design*, vol. 40, no. 1, pp. 88–115, Feb. 2012.
[20] X. Yin and S. Li, "Synthesis of Dynamic Masks for Infinite-Step Opacity," *IEEE Transactions on Automatic Control*, pp. 1–1, 2019.
[21] R. Liu, J. Lu, and C. N. Hadjicostis, "Opacity Enforcement via Attribute-Based Edit Functions in the Presence of an Intended Receiver," *IEEE Transactions on Automatic Control*, vol. 68, no. 9, pp. 5646–5652, Sep. 2023.
[22] R. Tai, L. Lin, Y. Zhu, and R. Su, "Privacy-preserving co-synthesis against sensor–actuator eavesdropping intruder," *Automatica*, vol. 150, p. 110860, Apr. 2023.
[23] A.-K. Schmuck, T. Moor, and R. Majumdar, "On the relation between reactive synthesis and supervisory control of non-terminating processes," *Discrete Event Dynamic Systems*, vol. 30, no. 1, pp. 81–124, Mar. 2020.
[24] O. Kupferman, O. Leshkowitz, and N. Shamash Halevy, "Synthesis with privacy against an observer," in *International Conference on Foundations of Software Science and Computation Structures*. Springer, 2024, pp. 256–277.
[25] Y. Wu and S. Lafortune, "Enforcement of opacity properties using insertion functions," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, Dec. 2012, pp. 6722–6728.
[26] Y. Ji, X. Yin, and S. Lafortune, "Opacity Enforcement Using Nondeterministic Publicly Known Edit Functions," *IEEE Transactions on Automatic Control*, vol. 64, no. 10, pp. 4369–4376, Oct. 2019.
[27] B. Chen, K. Leahy, A. Jones, and M. Hale, "Differential privacy for symbolic systems with application to Markov Chains," *Automatica*, vol. 152, p. 110908, Jun. 2023.
[28] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. Dray Jr, "Advanced encryption standard (AES)," 2001, updated in 2023.
[29] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, Mass: The MIT Press, 2008.
[30] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York, NY: Springer, 2008.
[31] A. Pnueli and R. Rosner, "Distributed reactive systems are hard to synthesize," in *FOCS*, Nov. 1990, pp. 746–757 vol.2.
[32] P. Madhusudan and P. Thiagarajan, "Distributed controller synthesis for local specifications," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, F. Orejas, P. Spirakis, and J. van Leeuwen, Eds. Springer Berlin, 2001, vol. 2076, pp. 396–407.
[33] K. Chatterjee, T. A. Henzinger, J. Otop, and A. Pavlogiannis, "Distributed synthesis for LTL fragments," in *2013 Formal Methods in Computer-Aided Design*. IEEE, 2013, pp. 18–25.
[34] B. Finkbeiner and S. Schewe, "Bounded synthesis," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 5, pp. 519–539, 2013.
[35] A.-K. Schmuck, T. Moor, and R. Majumdar, "On the relation between reactive synthesis and supervisory control of non-terminating processes," *Discrete Event Dynamic Systems*, vol. 30, no. 1, pp. 81–124, 2020.
[36] K. Inan, "Nondeterministic supervision under partial observations," in *11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*, ser. Lecture Notes in Control and Information Sciences, G. Cohen and J.-P. Quadrat, Eds. Berlin, Heidelberg: Springer, 1994, pp. 39–48.
[37] T.-S. Yoo and S. Lafortune, "Solvability of Centralized Supervisory Control Under Partial Observation," *Discrete Event Dynamic Systems*, vol. 16, no. 4, pp. 527–553, Dec. 2006.
[38] B. Finkbeiner, C. Hahn, J. Hofmann, and L. Tentrup, "Realizing omega-regular Hyperproperties," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, S. K. Lahiri and C. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 40–63.
[39] B. Finkbeiner, C. Hahn, P. Lukert, M. Stenger, and L. Tentrup, "Synthesizing Reactive Systems from Hyperproperties," in *Computer Aided Verification*, H. Chockler and G. Weissenbacher, Eds. Cham: Springer International Publishing, 2018, vol. 10981, pp. 289–306.
[40] B. Finkbeiner and N. Passing, "Compositional synthesis of modular systems," *Innovations in Systems and Software Engineering*, vol. 18, no. 3, pp. 455–469, Sep. 2022.
[41] B. Finkbeiner, N. Metzger, and Y. Moses, "Information flow guided synthesis," in *International Conference on Computer Aided Verification*. Springer, 2022, pp. 505–525.
[42] J. Chen, M. Ibrahim, and R. Kumar, "Quantification of Secrecy in Partially Observed Stochastic Discrete Event Systems," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 1, pp. 185–195, Jan. 2017.
[43] C. Keroglou and C. N. Hadjicostis, "Probabilistic system opacity in discrete event systems," *Discrete Event Dynamic Systems*, vol. 28, no. 2, pp. 289–314, Jun. 2018.
[44] E. Grädel, W. Thomas, and T. Wilke, *Automata, logics, and infinite games: a guide to current research*. Springer, 2003, vol. 2500.
[45] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal Logics for Hyperproperties," in *Principles of Security and Trust*, ser. Lecture Notes in Computer Science, M. Abadi and S. Kremer, Eds. Berlin, Heidelberg: Springer, 2014, pp. 265–284.
[46] P. Faymonville, B. Finkbeiner, and L. Tentrup, "Bosy: An experimentation framework for bounded synthesis," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 325–332.
[47] R. Meira-Góes, A. Wintenberg, S. Matsui, and S. Lafortune, "MDESops: an open-source software tool for discrete event systems modeled by automata," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 6093–6098, 2023.



Andrew Wintenberg (Graduate Student Member, IEEE) received his B.S. degree in Electrical Engineering and Mathematics from The University of Tennessee, Knoxville, USA, in 2018 and his M.S. and Ph.D. degrees in Electrical and Comnputer Engineering from the University of Michigan, Ann Arbor, in 2020 and 2024, respectively. His research interests include the safety and security of of discrete-event and cyberphysical systems through formal methods and abstraction.

Necmiye Ozay (Senior Member, IEEE) Necmiye Ozay received the B.S. degree from Bogazici University, Istanbul in 2004, the M.S. degree from the Pennsylvania State University, University Park in 2006 and the Ph.D. degree from Northeastern University, Boston in 2010, all in electrical engineering. She was a postdoctoral scholar at California Institute of Technology, Pasadena between 2010 and 2013. She is currently the Chen-Luan Family Faculy Development Professor of Electrical and Computer Engineering at University of Michigan, Ann Arbor, where she is also an associate professor of Electrical Engineering and Computer Science and of Robotics. Her research interests include control of dynamical systems, optimization, and formal methods with applications in cyber-physical systems, system identification, verification & validation, and autonomy.

Stéphane Lafortune (Life Fellow, IEEE) is the N. Harris McClamroch Collegiate Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, USA. He obtained his degrees from École Polytechnique de Montréal (B.Eng., 1980), McGill University (M.Eng., 1982), and the University of California at Berkeley (PhD, 1986), all in electrical engineering. He is a Fellow of IEEE (1999) and of IFAC (2017). His research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems. He co-authored, with Christos Cassandras, the textbook "Introduction to Discrete Event Systems - Third Edition" (Springer, 2021).