Enforcement of K-Step Opacity with Edit Functions

Andrew Wintenberg, Matthew Blischke, Stéphane Lafortune, and Necmiye Ozay

***Abstract*— Opacity is an information flow property for dynamic systems describing *plausible deniability*, that is whether an eavesdropper can deduce that "secret" behavior has occurred. In particular, K-step opacity considers secret actions that have occurred within the last K-steps in the past. We consider the problem of K-step opacity *enforcement* over automata using obfuscation. We present a general framework for K-step opacity enforcement and transform the problem of enforcing K-step opacity to enforcing current-state opacity. We can then apply existing obfuscation synthesis methods for current-state opacity to K-step opacity. We demonstrate this approach by enforcing privacy in the context of a novel contact tracing model.**

I. INTRODUCTION

Networked cyber-physical systems are becoming widespread throughout society in the form of autonomous vehicles, the smart grid, location-based services, and medical monitoring, to name but a few technological domains. These systems often transmit sensitive information across networks which can be compromised by a malicious eavesdropper. To address this concern, the use of formal methods has been proposed to understand information flow within these systems. In this work, we use the information flow property of *opacity* to express the privacy and security of cyber-physical systems. Opacity captures the notion of *plausible deniability*: opacity holds if sensitive or secret information of the system cannot be deduced by an outside eavesdropper. In other words, the system's *secret behavior* looks the same as its *nonsecret behavior*.

Opacity has gained considerable interest in the area of Discrete Event Systems (DES). Opacity has been studied using a variety of DES models like transition systems [1], Petri nets [2], finite state automata [3], and more. In order to accurately reflect the complex security requirements that may be required for these systems, a variety of notions of opacity have been formulated along with corresponding methods for verification. These notions include current-state opacity [3], initial-state opacity [4], language-based opacity [5], and approximate opacity [6]. Roughly, these notions differ in which behavior is considered secret and what capabilities are possessed by the eavesdropper. For example, current-state opacity requires that the eavesdropper cannot deduce that the system is currently at a secret state based on its observation of system events. While not revealing current secrets may sufficiently capture some security requirements, it is often the case that secrets of the past are sensitive as well. In this case, an eavesdropper can refine or *smooth* their knowledge of past secrets using current observations. To model this security concern, notions of K-step opacity [7], [8] have been proposed which require that the eavesdropper cannot deduce secrets within K subsequent observations. A thorough review of opacity in the context of DES, as of 2016, is provided in [9].

Oftentimes, it can be verified that an existing system is not opaque, but it is desired to *enforce* opacity upon the system. The problem of opacity enforcement has been approached using a variety of mechanisms in the literature. For example, supervisory control can enforce opacity by restricting behavior that would reveal secrets to an eavesdropper. The synthesis of such controllers has been investigated in [10], [11] for instance. However, it is not always feasible to alter existing system behavior, e.g., human behavior in a cyber-physical system. As an alternative to control, obfuscation can enforce opacity by altering the information ultimately available to the eavesdropper. Several works have investigated the synthesis of *edit functions* which selectively insert and delete outputs from the system [12], [13]. These approaches synthesize edit functions, which effectively enforce current-state opacity, as winning strategies to finite two-player reachability games as in [14]. Related obfuscation strategies, such as delaying observations at runtime [8] and dynamic masks [15], have also been studied.

In this paper, we consider the problem of synthesizing edit functions to enforce K-step opacity. To the best of our knowledge, this problem has not yet been explicitly considered in the literature. As noted in [12], synthesis methods for current-state opacity can be applied to enforce other notions of opacity that can be transformed into current-state opacity, provided the resulting enforcement strategy for current-state opacity can be mapped back to the original notion of opacity under consideration. For example, it was found in [16] that initial-state and language-based opacity can be transformed into current-state opacity. Recently, a novel language-based formulation of the various notions of K-step and infinite step opacity was established, for the first time, in [17]. While [17] focuses on the verification of K-step opacity over finite automata, in this work we focus on the enforcement of K-step opacity with edit functions. We show how safe edit functions can be synthesized by suitably leveraging existing methods for current-state opacity. We demonstrate this approach on a case study by synthesizing edit functions enforcing location privacy on a system modeling the motion of individuals using a contact-tracing app whose data is available to a malicious eavesdropper.

This work was supported in part by NSF grants CNS-1738103, CNS-1801342, and ECCS-1553873.

A. Wintenberg, M. Blischke, S. Lafortune, and N. Ozay are with the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, MI 48109, United States, email: {awintenb, matblisc, stephane, necmiye}@umich.edu

The remaining sections of this paper are organized as follows. A review of basic automata theory and a general framework for opacity including K-step opacity is presented in Section II. Section III discusses how edit functions can be used as a means of obfuscation to enforce K-step opacity. Synthesis methods for such edit functions are presented in Section IV which leverage existing methods for current-state opacity. Section V demonstrates this approach to preserve privacy in a system modeling contact-tracing. Finally, the paper is concluded in Section VI.

II. PRELIMINARIES

In this section we review basic automata theory and summarize the general framework for opacity from [17] along with results for expressing K-step opacity in this setting.

A. Operations on automata

A nondeterministic finite automaton (NFA) is defined by a tuple $G = (Q, E, f, Q_0, Q_m)$ with a finite set of states Q, events E, transition function $f : Q \times (E \cup \{\epsilon\}) \to 2^Q$, initial states Q_0 and marked states Q_m. We also extend f to the domain $Q \times E^*$ in the standard way. Here ϵ denotes the empty string. A nonempty deterministic finite automaton is a special kind of NFA where $|Q_0| = 1$ and $|f(q, e)| \leq 1$ for all $q \in Q$ and $e \in E$. We denote the set of finite strings over the alphabet E as E^*. We denote the set of prefixes of a string $s \in E^*$ as $\overline{s}$ and the prefixes of the language $L \subseteq E^*$ as $\overline{L}$. We denote the length of a string s as $|s|$. The language generated by G is defined by

$$\mathcal{L}(G) = \{s \in E^* \mid \exists q_0 \in Q_0 \ f(q_0, s) \neq \emptyset\}. \tag{1}$$

Likewise the language marked by G is defined by

$$\mathcal{L}_m(G) = \{s \in E^* \mid \exists q_0 \in Q_0 \ \exists q_m \in Q_m \ q_m \in f(q_0, s)\}. \tag{2}$$

For automata G and H, we write $G \times H$ for the synchronous product of G and H, and $\det(G)$ for the determinization of G using the standard power set construction. We also use the notation $P_{E_o} : E^* \to E_o^*$ to denote the projection of strings with respect to observable events $E_o \subseteq E$. These notions are defined in detail in [18].

B. Problem setting & opacity framework

We consider discrete event systems modeled by a nondeterministic automaton $\mathcal{A} = (X, \Sigma, \delta, X_0)$ without marked states. Marked states are not considered as we are concerned with the prefix-closed behavior of the system. The *secret behavior* of this system is defined in terms of a subset of the system's states $X_S \subseteq X$ which are called *secret*. This set is encoded with a state-labeling map $\ell : X \to A$ where $A = \{\mathrm{S}, \mathrm{NS}\}$ defined for $x \in X$ by $\ell(x) = S$ if $x \in X_\mathrm{S}$ and $\ell(x) = \mathrm{NS}$ otherwise. By viewing events as inputs and state labels as outputs, the behavior or runs of $\mathcal{A}$ under ℓ can be described as sequences of input-output pairs[1]. In order to carry the label of the initial state in a run, the artificial event σ_{init} disjoint from Σ is introduced. The set of input-output pairs is denoted by $E = (\Sigma \cup \{\sigma_{init}\}) \times A$. We can transform $\mathcal{A}$ into an automaton generating strings of input-output pairs by augmenting its transitions with the state labels of their destination. Additionally, an artificial initial state x_{init} disjoint from X is introduced as a source for σ_{init} events.

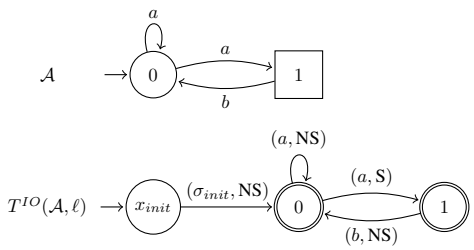

Fig. 1. An automaton $\mathcal{A}$ (top) where the square state 1 is considered secret so $\ell(0) = \mathrm{NS}$ and $\ell(1) = S$. The label-transform $T^{IO}(\mathcal{A}, \ell)$ (bottom) recognizes the input-output sequences of $\mathcal{A}$ under ℓ.

Definition 2.1: The *label-transform* of an NFA $\mathcal{A} = (X, \Sigma, \delta, X_0)$ under the label map $\ell : X \to A$, is an NFA $T^{IO}(\mathcal{A}, \ell) = (Q, E, f, Q_0, Q_m)$, where $Q = X \cup \{x_{init}\}$, $E = (\Sigma \cup \{\sigma_{init}\}) \times A$, $Q_0 = \{x_{init}\}$, $Q_m = X$, and whose transitions are defined for all $x \in X, \sigma \in \Sigma, a \in A$ by

$$\begin{aligned} f(x_{init}, (\sigma_{init}, a)) &= \{x_0 \in X_0 \mid \ell(x_0) = a\}, \\ f(x, (\sigma, a)) &= \{x' \in \delta(x, \sigma) \mid \ell(x') = a\}. \end{aligned} \tag{3}$$

This label-transform is similar to the transform of automata with state outputs to input-output automata in [18]. An example of this label-transform is depicted in Figure 1. The input-output behavior is then defined as follows.

Definition 2.2: The *set of input-output sequences* of an NFA $\mathcal{A} = (X, \Sigma, \delta, X_0)$ under the label map $\ell : X \to A$ is defined as

$$\mathcal{L}^{IO}(\mathcal{A}, \ell) = \mathcal{L}_m(T^{IO}(\mathcal{A}, \ell)). \tag{4}$$

We denote the behavior or set of runs of the system $\mathcal{A}$ under ℓ as $R = \mathcal{L}^{IO}(\mathcal{A}, \ell)$. The information flow out of the system is modeled through an *observation map* $\Theta : R \to O$ mapping runs in R to observations in the space O. We consider an eavesdropper attempting to deduce whether or not certain secret behaviors have occurred using these observations. We assume that the eavesdropper *a priori* knows the system's behavior R and the original observation map Θ. We assume these observations are produced by the projection onto a set of observable events $\Sigma_o \subseteq \Sigma$. While the framework of [17] considers σ_{init} to be observable, for simplicity in discussing obfuscation we take σ_{init} to be unobservable. We assume the observation map $\Theta : R \to O$ is induced by Σ_o

[1]Using the language of $\mathcal{A}$ alone to describe its behavior is not sufficient for K-step opacity. As noted by [7], a single string can be considered both secret and nonsecret depending on the state path taken. To emphasize this point, we denote automata representing state-based behavior as $\mathcal{A}$ and language-based behavior as G.

so that $O = \Sigma_o^*$ and for a run $r = (\sigma_0, a_0) \cdots (\sigma_n, a_n) \in R$ it holds that

$$\Theta(r) = P_{\Sigma_o}(\sigma_0 \cdots \sigma_n)\,. \tag{5}$$

Then for any NFA $G = (Q, E, f, Q_0, Q_m)$, we can construct an NFA $\Theta(G)$ by replacing unobservable events $\Sigma \setminus \Sigma_o$ of G with the empty string ϵ. The NFA $\Theta(G)$ encodes the observations of G in that

$$\mathcal{L}(\Theta(G)) = \Theta(\mathcal{L}(G)), \quad \mathcal{L}_m(\Theta(G)) = \Theta(\mathcal{L}_m(G))\,. \tag{6}$$

Under the normal assumption that the automaton $\mathcal{A}$ has a nonempty set of initial states[2], it holds for $R = \mathcal{L}^{IO}(\mathcal{A}, \ell)$ that

$$\Theta(R) = \mathcal{L}(\Theta(T^{IO}(\mathcal{A}, \ell)))\,. \tag{7}$$

Various notions of opacity relate observations of *secret and nonsecret behavior*. For example, current-state opacity (CSO) requires that every run should have the same observation as a run not ending at a secret state in X_{S}. In this case, the eavesdropper cannot determine the system is currently at a secret state. Formally, given

$$R_{\mathrm{NS}} = \{(\sigma_0, a_0) \cdots (\sigma_n, a_n) \in R \mid a_n = \mathrm{NS}\}\,, \tag{8}$$

the system $\mathcal{A}$ with secret states X_S is said to be *current-state opaque* if $\Theta(R) \subseteq \Theta(R_{\mathrm{NS}})$.

C. K-step Opacity

In a dynamic system, security may require not revealing secrets of the past as well as the present. In this case, CSO is not sufficient for describing security. A stronger notion, K-step opacity, requires that secrets are not revealed within K observations of their occurrence for $K \in \mathbb{N} = \{0, 1, \cdots\}$. While CSO can be described with one class of secret behavior, i.e., runs ending at a secret state, K-step opacity can more easily be described in terms of multiple classes of secret behavior. These classes correspond to secrets occurring k observations ago for $k \in \{0, \cdots, K\}$. Two types of secrets between consecutive observations are considered. In the first type, called **type 1**, *at least one secret state is visited*. In the second type, called **type 2**, *only secret states are visited*. Given $k \in \mathbb{N}$ and $j \in \{1, 2\}$, the *k-delayed type j nonsecret behavior* describes runs which did not exhibit type j secrets between $k+1$ and k observations ago. For an automaton $\mathcal{A}$, secret label map ℓ, and observable events Σ_o, this behavior can be specified using an automata $H_{\mathrm{NS},j}(k)$ over the set of input-output pairs E as presented in [17] . Formally the k-delayed type j nonsecret behavior $R_{\mathrm{NS},j}(k)$ is given by

$$R_{NS,j}(k) = \mathcal{L}_m(T^{IO}(\mathcal{A}, \ell) \times H_{NS,j}(k)) \subseteq R\,. \tag{9}$$

Remark 1: It should also be noted that the construction of the nonsecret specification automata $H_{\mathrm{NS},j}(k)$ depends on the secret label map ℓ and the set of observable events. While in this work, we use the convention that σ_{init} is unobservable with respect to the observation map Θ induced by the observable events Σ_o, in accordance with [17] the nonsecret specification automaton $H_{\mathrm{NS},j}(k)$ is constructed with $\Sigma_o \cup \{\sigma_{init}\}$ as the set of observable events. This construction specifies the appropriate k-delayed nonsecret behavior, while the new convention for σ_{init} simplifies our presentation of enforcement by ensuring the set $\Theta(R)$ is prefixed-closed as seen in equation (7). For example consider $H_{\mathrm{NS},2}(2)$ depicted in Figure 2. Strings in $\tilde{E}_o^2$ are accepted starting at state $(1,0)$ while strings in $\tilde{E}_o^3$ might not be. This is because a run of the system G with one event (after the artificial initial event) is trivially 2-delayed nonsecret behavior while a run with two events is not if the initial state is secret.

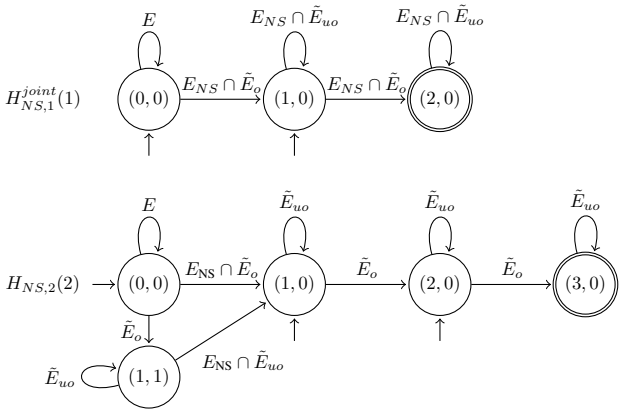

Fig. 2. The nonsecret specification automata $H_{\mathrm{NS},1}^{joint}(1)$ and $H_{\mathrm{NS},2}(2)$. These automata are defined over the input-output pairs $E = (\Sigma \cup \{\sigma_{init}) \times \{\mathrm{S}, \mathrm{NS}\}$. Here $E_{NS} = (\Sigma \cup \{\sigma_{init}\}) \times \{NS\}$. Likewise $\tilde{E}_o = (\Sigma_o \cup \{\sigma_{init}\}) \times \{\mathrm{S}, \mathrm{NS}\}$ and $\tilde{E}_{uo} = E \setminus \tilde{E}_o$.

Two distinct notions of K-step opacity arise by combining these classes of nonsecret behavior. An eavesdropper whose goal is to deduce *when* a secret occurred considers these classes *separately*.

Definition 2.3: For $K \in \mathbb{N}$ and $j \in \{1, 2\}$, the system $\mathcal{A}$, with secret label map ℓ, and observable events Σ_o is *separately K-step opaque with type j secrets* if

$$\forall k \in \{0, \cdots, K\},\ \Theta(R) \subseteq \Theta(R_{\mathrm{NS},j}(k))\,, \tag{10}$$

or equivalently

$$\Theta(R) \subseteq \bigcap_{k=0}^{K} \Theta(R_{\mathrm{NS},j}(k))\,. \tag{11}$$

Alternatively, an eavesdropper whose goal is to deduce *if* a secret occurred considers these classes *jointly*.

Definition 2.4: For $K \in \mathbb{N}$ and $j \in \{1, 2\}$, the system $\mathcal{A}$, with secret label map ℓ, and observable events Σ_o is *jointly K-step opaque with type j secrets* if

$$\Theta(R) \subseteq \Theta\left(\bigcap_{k=0}^{K} R_{\mathrm{NS},j}(k)\right)\,. \tag{12}$$

It is shown in [17] that these definitions capture previous notions of K-step opacity. Namely, joint K-step opacity with type 1 secrets is equivalent to strong K-step opacity as

[2]This assumption guarantees that either $(\sigma_{init}, \mathrm{S})$ or $(\sigma_{init}, \mathrm{NS})$ is an element of $R = \mathcal{L}_m(T^{IO}(\mathcal{A}, \ell))$ so that $\epsilon \in \Theta(R)$ under the convention that σ_{init} is unobservable for Θ.

defined in [8]. Likewise, separate K-step opacity with type 2 secrets is equivalent to the original definition of K-step opacity in [3] (called weak K-step opacity in [8]).

In order to concisely represent the intersection of the delayed behavior for joint K-step opacity, automata $H^{joint}_{\text{NS},j}(K)$ are also presented in [17] such that

$$\mathcal{L}_m(T^{IO}(\mathcal{A},\ell) \times H^{joint}_{NS,j}(K)) = \bigcap_{k=0}^{K} R_{NS,j}(k)\,. \tag{13}$$

For example $H^{joint}_{\text{NS},1}(1)$ is depicted in Figure 2. Also the nonsecret specification automata $H_{\text{NS},j}(K)$ and $H^{joint}_{\text{NS},j}(K)$ are complete so that

$$\mathcal{L}(H_{\text{NS},j}(K)) = \mathcal{L}(H^{joint}_{\text{NS},j}(K)) = E^*\,. \tag{14}$$

III. ENFORCEMENT OF K-STEP OPACITY WITH OBFUSCATION

In this section, we discuss how obfuscation in the form of edit functions can be used to enforce K-step opacity over automata. We describe notions of the safety of edit functions for enforcing K-step opacity when the edit function is either known or unknown to the eavesdropper.

A. Obfuscation Model

Recall that opacity is violated when the eavesdropper observes information output by the system that reveals secret behavior. This situation can be avoided with *obfuscation*, i.e., altering the information output by the system to fool the eavesdropper while maintaining the system's utility. We model this type of obfuscator as a deterministic edit function as in [19] which selectively deletes events output by the system and possibly inserts fictitious ones. We represent edit functions as a mapping from strings output by the system to strings observed by the eavesdropper.

Definition 3.1: A *deterministic edit function* over the space $O = \Sigma_o^*$ is a function $M : O \to O$ such that

$$\forall o \in O,\ \forall o' \in \overline{o},\ M(o') \in \overline{M(o)}\,. \tag{15}$$

This condition ensures that future outputs are consistent with previous ones. Such an edit function can be implemented in many ways. For example under some regularity assumptions, an edit function can be represented by a finite state transducer as in [20].

Consider the system from Section II consisting of a system $\mathcal{A}$ with secret label map ℓ and observable events Σ_o. Recall the behavior of this system is given by $R = \mathcal{L}^{IO}(\mathcal{A},\ell)$ which is mapped to observations by the map $\Theta : R \to O$ induced by the observable events Σ_o with $O = \Sigma_o^*$. We can compose an edit function $M : O \to O$ with the *original* observation map Θ to construct the *obfuscated* observation map $M \circ \Theta : R \to O$. We call the set $\Theta(R)$ the *original observations* of the system behavior. As an edit function may insert a sequence of events, we consider the *obfuscated observations* of system behavior to be the prefix-closed set $\overline{M(\Theta(R))}$. To enforce opacity, an edit function must guarantee that its obfuscated observations are contained in a set of *safe* observations that do not reveal secrets to the eavesdropper. Figure 3 depicts the flow of information we consider.

B. Notions of opacity under enforcement

While we assume the eavesdropper knows how information is originally output from the system through the map Θ, we can consider different levels of knowledge the eavesdropper has about the edit function M. These correspond to different levels of *safety* ensured by the enforcement mechanism. If the edit function M is unknown to the eavesdropper or *private*, then it may be desired that obfuscated observations be consistent with the original observations of nonsecret behavior.

Definition 3.2: For $K \in \mathbb{N}$, $j \in \{1,2\}$, we say an edit function M is privately safe for separate K-step opacity with type j secrets if

$$\overline{M(\Theta(R))} \subseteq \bigcap_{k=0}^{K} \Theta(R_{\text{NS},j}(k))\,. \tag{16}$$

Definition 3.3: For $K \in \mathbb{N}$, $j \in \{1,2\}$, we say an edit function M is privately safe for joint K-step opacity with type j secrets if

$$\overline{M(\Theta(R))} \subseteq \Theta\left(\bigcap_{k=0}^{K} R_{\text{NS},j}(k)\right)\,. \tag{17}$$

Alternatively, if the edit function M is known to the eavesdropper or *public*, then the obfuscated observations of system behavior should be consistent with the obfuscated observations of nonsecret behavior. In this work, we focus on privately safe edit functions. These notions of public and private safety of edit functions for K-step opacity (joint or separate, type 1 or 2 secrets) mirror the definitions for the case of CSO from [19].

Remark 2: While we only consider deterministic edit functions, these definitions and the following synthesis approaches can be generalized to nondeterministic edit functions which are strictly more powerful than deterministic ones [19]. Additionally, a similar approach can be used to define public and private safety for opacity enforcement with supervisory control.

IV. SYNTHESIS OF EDIT FUNCTIONS

In this section, we describe how to synthesize edit functions enforcing private safety for joint and separate K-step opacity, as in Definitions 2.4 and 2.3, using existing methods for CSO such as [13], [21], [22]. Abstracting away details specific to each implementation, these synthesis methods can broadly be described as solving the following problem.

Problem 1: Given an NFA $G = (Q, \Sigma_o, f, Q_0, Q_m)$ and a class of edit functions $\mathcal{M} \subseteq \{M : \mathcal{L}(G) \to \Sigma_o^*\}$, find an edit function $M \in \mathcal{M}$ such that

$$\overline{M(\mathcal{L}(G))} \subseteq \mathcal{L}_m(G)\,. \tag{18}$$

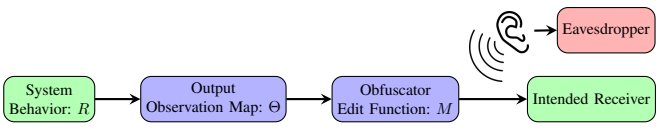

Fig. 3. System framework for enforcement of opacity with edit functions.

In this problem, the marked states Q_m encode admissible or nonsecret states, and unobservable events are represented with ϵ transitions. The set $\mathcal{M}$ describes the edit functions that satisfy additional constraints that can be imposed in a given synthesis method. For example, the methods of [13], [21] synthesize edit functions with a bound on the number of consecutive insertions. After determinizing G, the methods of [22] also consider *utility constraints* which limit the difference between original observations and their obfuscations.

Remark 3: Other mechanisms for enforcement, such as the dynamic masks considered in [15], can be viewed as a specific type of edit function. Additionally while Problem 1 describes private safety for CSO, there is a similar problem for public safety with corresponding synthesis methods that could be applied to K-step opacity. For example synthesis of edit functions which are both publicly and privately safe is discussed in [21] while the R-enforcers of [8] can be viewed as edit functions enforcing public safety using delay.

Given an automaton $\mathcal{A}$ with secret label map ℓ and observable events Σ_o, we can use these synthesis methods to enforce joint or separate K-step opacity with type j secrets by applying the methods to an appropriately constructed automaton G as in Problem 1.

A. Enforcing Joint K-step Opacity

As joint K-step opacity with type j secrets is defined in terms of only one class of nonsecret runs, namely the intersection of the sets $R_{\text{NS},j}(k)$ for $k \in \{0, \cdots, K\}$, we can apply synthesis methods for current-state opacity directly.

Theorem 4.1: Let $G = \Theta(T^{IO}(\mathcal{A}, \ell) \times H^{joint}_{\text{NS},j}(K))$. An edit function $M \in \mathcal{M}$ is privately safe for joint K-step opacity with type j secrets over $\mathcal{A}$, ℓ, Σ_o if and only if it is a solution to Problem 1 for G, $\mathcal{M}$.

Proof: From equations (7) and (14) it holds that

$$\begin{aligned} \mathcal{L}(G) &= \Theta(\mathcal{L}(T^{IO}(\mathcal{A}, \ell)) \cap \mathcal{L}(H^{joint}_{\text{NS},j}(K))) \\ &= \Theta(R \cap E^*) = \Theta(R)\,. \end{aligned} \tag{19}$$

Likewise from equations (5) and (13)

$$\begin{aligned} \mathcal{L}_m(G) &= \Theta(\mathcal{L}_m(T^{IO}(\mathcal{A}, \ell) \times H^{joint}_{\text{NS},j}(K))) \\ &= \Theta\left(\bigcap_{k=0}^{K} R_{\text{NS},j}(K)\right)\,. \end{aligned} \tag{20}$$

So solutions to Problem 1 satisfying (18) are exactly the edit functions which satisfy (17) in Definition 3.3 for private safety for joint opacity. ∎

If a deterministic automaton is required, the same result holds for the determinization $\det(G)$ instead of G. Here the automaton $\det(G)$ corresponds to the secret observer automaton G_{SO} for joint opacity from [17]. So to enforce K-step opacity for an automaton $\mathcal{A}$, we construct G as in Theorem 4.1, synthesize an edit function M as a solution to Problem 1 for G, and apply M directly to the outputs of the original system. This is possible as the language of G is the set of observations produced by the original automaton $\mathcal{A}$. We demonstrate this in the following example.

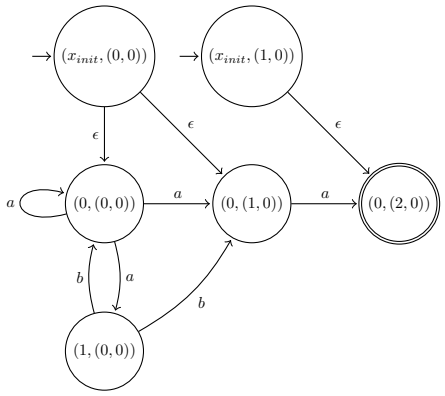

Fig. 4. The automaton $G = \Theta(T^{IO}(\mathcal{A}, \ell) \times H^{joint}_{\text{NS},1}(1))$ in Example 4.1.

Example 4.1: In this example we construct an edit function enforcing joint 1-step opacity with type 1 secrets. Consider the automaton $\mathcal{A}$, secret label map ℓ, and their label-transform $T^{IO}(\mathcal{A}, \ell)$ depicted in Figure 1. We assume that all events of $\mathcal{A}$ are observable so $\Sigma_o = \{a, b\}$. Consider the nonsecret specification automaton $H^{joint}_{\text{NS},1}(1)$ as depicted in Figure 2 constructed for $\mathcal{A}$, ℓ, and Σ_o. This automaton marks sequences of input-output pairs corresponding to runs that have not visited a secret state since 1 observation ago. We construct the automaton $G = \Theta(T^{IO}(\mathcal{A}, \ell) \times H^{joint}_{\text{NS},1}(1))$ which is depicted in Figure 4. This automaton G marks the observations of the behavior of $\mathcal{A}$ that do not violate joint 1-step opacity with type 1 secrets. To better understand these observations we construct $\det(G)$ which is language equivalent and is depicted in Figure 5. We can see that the observation ab is not marked and hence is unsafe. This is because if the eavesdropper observes the event b, they reason that the system was in the secret state 1 exactly 1 step ago.

If knowing the total number of events executed in the system is important, we can consider the following class of edit functions

$$\mathcal{M} = \{M : \mathcal{L}(G) \to \Sigma_o^* \mid \forall s \in \mathcal{L}(G),\ |M(s)| = |s|\}\,. \tag{21}$$

Then a solution to Problem 1 for G, $\mathcal{M}$ is the edit function M which replaces all occurrences of b with a so $\forall s \in \mathcal{L}(G),\ M(s) = a^{|s|}$. We can verify that M is a solution to Problem 1 as the set of obfuscated observations is $\overline{M(\Theta(\mathcal{L}(G))} = \{a\}^*$ which is safe. Thus by Theorem 4.1, the edit function M is privately safe for 1-step opacity with type j secrets. This makes sense: M hides the only event b that would reveal the secret to the eavesdropper.

B. Separate Case

Enforcing separate K-step opacity is complex as we must consider multiple classes of nonsecret behavior $R_{\text{NS},j}(k)$ for $k \in \{0, \cdots K\}$ and type j secrets. However, like it suffices to consider observations of the intersection of these nonsecret behaviors $\Theta\left(\bigcap_{k=0}^{K} R_{\text{NS},j}(k)\right)$ for joint K-step opacity, it suffices to consider the intersection of the observations of these nonsecret behaviors $\bigcap_{k=0}^{K} \Theta(R_{\text{NS},j}(k))$ for separate

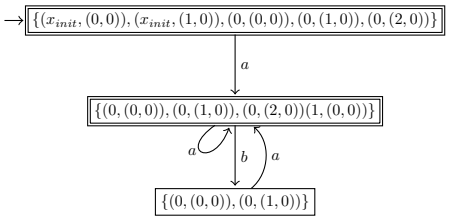

Fig. 5. The automaton $\det(G)$ from Example 4.1.

K-step opacity in terms of private safety. While we could represent this latter set of observations with the automaton

$$\prod_{k=0}^{K} \Theta(T^{IO}(\mathcal{A},\ell) \times H_{\text{NS},j}(k))\,, \tag{22}$$

we can avoid this product by observing that each $H_{\text{NS},j}(k)$ is contained within $H_{\text{NS},j}(K)$ for $k \leq K$, i.e, they are subautomata. Specifically as shown in [17], for each $k \in \{0,\cdots,K\}$ there exists a set $Q_{m,H,k}$ of states of $H_{\text{NS},j}(K)$ such that the language of $H_{\text{NS},j}(K)$ marked by the states $Q_{m,H,k}$ is $\mathcal{L}_m(H_{\text{NS},j}(k))$. For example in the automaton $H_{\text{NS},2}(2)$ depicted in Figure 2, these sets are given by $Q_{m,H,0} = \{(1,0)\}$, $Q_{m,H,0} = \{(2,0)\}$, $Q_{m,H,0} = \{(3,0)\}$.

Theorem 4.2: Let $G = \det(T^{IO}(\mathcal{A},\ell) \times H_{\text{NS},j}(K))$ using the power set construction for determinization. Let the marked states Q_m of G be redefined as

$$Q_m = \{\overline{q} \mid \forall k \in \{0,\cdots,K\}\ \exists q \in \overline{q} \cap Q \times Q_{m,H,k}\}\,. \tag{23}$$

Then an edit function $M \in \mathcal{M}$ is privately safe for separate K-step opacity with type j secrets over $\mathcal{A}$, ℓ, Σ_o if and only if it is a solution to Problem 1 for G, $\mathcal{M}$.

Proof: From equations (7) and (14) it holds that

$$\begin{aligned}\mathcal{L}(G) &= \Theta(\mathcal{L}(T^{IO}(\mathcal{A},\ell)) \cap \mathcal{L}(H_{\text{NS},j}(K))) \\ &= \Theta(R \cap E^*) = \Theta(R)\,.\end{aligned} \tag{24}$$

Likewise from equations (5) and (9)

$$\begin{aligned}\mathcal{L}_m(G) &= \bigcap_{k=0}^{K} \Theta(\mathcal{L}_m(T^{IO}(\mathcal{A},\ell \times H_{\text{NS},j}(k)))) \\ &= \bigcap_{k=0}^{K} \Theta(R_{\text{NS},j}(k))\,.\end{aligned} \tag{25}$$

So solutions to Problem 1 satisfying (18) are exactly the edit functions which satisfy (16) in Definition 3.2 for private safety for joint opacity. ∎

The automaton G in this theorem corresponds to the secret observer automaton G_{SO} for separate opacity from [17]. We can then synthesize edit functions by solving Problem 1 as in the joint case.

V. CONTACT TRACING EXAMPLE

In this section, we demonstrate an application of K-step opacity enforcement in the context of contact tracing smartphone apps, developed in response to the COVID-19 pandemic, that record proximity between users to a centralized server. These apps raise a number of privacy concerns as described in [23], [24]. We propose the use of obfuscation to enforce privacy for users of these apps while maintaining the utility of these apps for public health. We demonstrate this through a small example where a malicious user who gains access to this information may be able to combine it with partial location information to determine that another user has visited some secret location. We then show how to enforce K-step opacity with a privately safe edit function using our approach, specifically, by leveraging Theorem 4.1. This edit function was synthesized with the methods from [22] as implemented in the *EdiSyn* library[3]. The automata constructions presented in this work and an interface to the EdiSyn library are provided by the *DESops* library[4].

A. Preliminaries

Our model assumes that there are some number of users, including one user that acts as a malicious eavesdropper, that can move freely between a shared set of locations. We suppose that the malicious user has full knowledge of their own movements, in addition to partial knowledge of the other users' movements based on a partition of the map into discrete regions. This partial knowledge is modeled similarly to the automaton model for location-based services described in [25], but with event labels based on the destination node instead of the source node. We further assume that the malicious user has access to information from the contact tracing app from which they can deduce the sizes of contact clusters that exist at any given time, where we consider a contact cluster to be a set of multiple users sharing the same location. The malicious eavesdropper does not know the locations of the contact clusters, or which users are in which contact clusters.

Our procedure requires as input a simple undirected graph $\Gamma = (\mathcal{V}, \mathscr{E})$ that represents locations by the set of vertices $\mathcal{V}$, and the physical paths between locations by the set of edges $\mathscr{E}$. It also requires a partition P of $\mathcal{V}$ into regions, a fixed number of users n, and some definition of secret behavior. We additionally define **region** : $\mathcal{V} \to P$ so that for $v \in \mathcal{V}$, **region**(v) is the region containing v.

Throughout this section, we illustrate our procedure using Γ as shown in Figure 6, with $P = \{\mathcal{R}, \mathcal{T}\}$ as shown. We also suppose that $n = 3$, and user 2 visiting location 5 as the secret state of the system.

B. Mobility Model

Using Γ, we first contruct the map automaton $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Delta, \mathcal{V}_0)$, where $\mathcal{V}$ is the set of states, $\mathcal{E}$ is the set of events, $\Delta : \mathcal{V} \times \mathcal{E} \to 2^{\mathcal{V}}$ is the transition function, and $\mathcal{V}_0$ is the set of initial states. We define $\mathcal{V}_0 = \mathcal{V}$, $\mathcal{E} = \{\tau\}$, and $\Delta(v,\tau) = \{v\} \cup \{u \in \mathcal{V} : (u,v) \in \mathscr{E}\}$ for all $v \in \mathcal{V}$.

For each $i \in \{1,2,\ldots,n\}$, we construct an individual automaton $\mathcal{G}_i$, which represents all movements that user i is allowed to make in a single time step. We construct $\mathcal{G}_1$

[3] https://gitlab.eecs.umich.edu/M-DES-tools/EdiSyn
[4] https://gitlab.eecs.umich.edu/M-DES-tools/desops

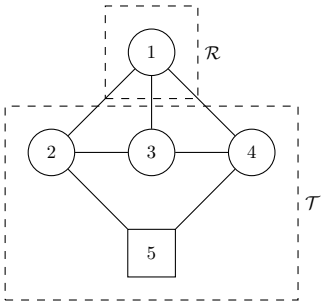

Fig. 6. The graph Γ that represents the locations and the physical paths between them. $\mathcal{R}$ and $\mathcal{T}$ represent the partition of locations into distinct regions. Location 5 is considered to be secret.

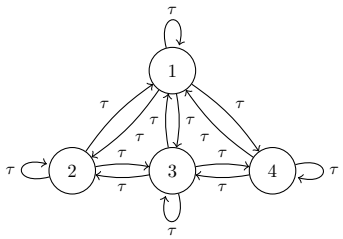

Fig. 7. The individual automaton $\mathcal{G}_1$, representing all possible movements by the malicious user in a single time step. The individual automata $\mathcal{G}_2$ and $\mathcal{G}_3$ are similar, but also include the secret state 5.

from $\mathcal{G}$ by removing the secret location, since we assume the malicious user is not allowed to enter the secret location. For $i > 1$, we let $\mathcal{G}_i = \mathcal{G}$, since other users are unrestricted in their movement. The individual automaton $\mathcal{G}_1$ for our example is shown in Figure 7. The individual automata $\mathcal{G}_2$ and $\mathcal{G}_3$ are not shown, but are similar with $\mathcal{G}_1$, with the additional inclusion of the secret location 5. We then construct $\mathcal{H} = \mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_n$, which gives the full model of all possible movements by all users within the system. The state set of $\mathcal{H}$ is then $\mathcal{V}^n$, and for each state $x \in \mathcal{V}^n$ with $x = (v_1, ..., v_n)$, we have that v_i is the location of user i.

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

$$\alpha(x) = (\alpha_1, ..., \alpha_n) \tag{26}$$

where for each i,

$$\alpha_i = \begin{cases} v_1, & i = 1 \\ \textbf{region}(v_i), & i > 1. \end{cases} \tag{27}$$

The result is that $\alpha(x)$ denotes the locations of each user, as observed by the malicious eavesdropper, when the system is in state x. We also define the multiset

$$\beta(x) = \{|\mathcal{B}_v| : |\mathcal{B}_v| > 1\} \tag{28}$$

where for each $v \in \mathcal{V}$,

$$\mathcal{B}_v = \{i \in \{1, \ldots, n\} : v_i = v\}. \tag{29}$$

The result is that $\beta(x)$ represents the sizes of the contact clusters that exist when the system is in state x. Note that in general, $\beta(x)$ must be a multiset since there may be more than one contact cluster of a given size, but since also the contact clusters are indistinguishable and thus their sizes should be unordered. For our example with $n = 3$, we note that $\beta(x)$ has only three possible values: $\beta(x) = \varnothing$ if all states in x are distinct, $\beta(x) = \{2\}$ if two states of x are identical and the third is distinct, or $\beta(x) = \{3\}$ if all states of x are identical.

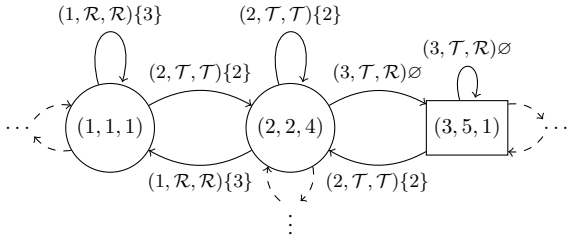

Fig. 8. A small subautomaton of $\mathcal{A}$. Events are labeled according to their target state. The state $(3, 5, 1)$ is secret since $v_2 \in \mathcal{V}_S = \{5\}$.

Finally, we construct $\mathcal{A}$ from $\mathcal{H}$ by relabeling each transition as the concatenation $\alpha(x)\beta(x)$ where x is the state at which the transition ends, and α and β are the location and contact markings as defined in equations (26) and (28). We let the observable event set of $\mathcal{A}$ be $\Sigma_o = \Sigma$. We additionally mark states with $v_2 \in \mathcal{V}_S$ as secret. For our example, $\mathcal{A}$ contains 100 states and 5,054 transitions, and so is unable to be shown in full. However, a small subautomaton of $\mathcal{A}$ is shown in Figure 8, illustrating the result of the event relabeling.

C. Opacity Enforcement

Using existing methods of opacity verification, we can determine that the system is current-state opaque, but that it is neither separately nor jointly 1-step opaque. Note that since $\Sigma_o = \Sigma$, there is no distinction between type 1 or type 2 secrets. One event sequence that violates 1-step opacity is

$$o = \Big((1,\mathcal{T},\mathcal{R})\{2\},\ (3,\mathcal{T},\mathcal{T})\varnothing,\ (4,\mathcal{T},\mathcal{T})\{3\}\Big), \tag{30}$$

which corresponds to the state-estimate sequence

$$\begin{aligned} X_1 &= \{(1,2,1),(1,3,1),(1,4,1),(1,5,1)\}, \\ X_2 &= \{(3,2,4),(3,5,4),(3,4,2),(3,5,2)\}, \\ X_3 &= \{(4,4,4)\}. \end{aligned} \tag{31}$$

Each X_i contains at least one nonsecret state, and thus current-state opacity is not violated by o. However, the only state in X_2 from which the event $o_3 = (4,\mathcal{T},\mathcal{T})\{3\}$ can occur is the secret state $(3, 5, 4)$, and thus 1-step opacity is violated.

To enforce joint 1-step opacity, we first use $\mathcal{A}$ to construct G as defined in Theorem 4.1, with $K = 1$. We additionally constrain the set $\mathcal{M}$ of allowable edit functions in two ways. First, we only allow the edit function to replace events, i.e. deleting events entirely or inserting new events where none previously existed is not allowed. Second, we enforce a utility constraint as is defined in [22]. This constraint prevents the obfuscator from changing any of the location

information observed by the eavesdropper directly. To construct this constraint, we first define the utility distance $D_{\mathcal{A}} : \mathcal{V}^n \times \mathcal{V}^n \to \{0, 1\}$ over the automaton $\mathcal{A}$ such that for $x, y \in \mathcal{V}^n$, we have

$$D_{\mathcal{A}}(x, y) = \begin{cases} 0, & \alpha(x) = \alpha(y) \\ 1, & \text{otherwise.} \end{cases} \tag{32}$$

where α is as defined in equation (26). We then transform $D_{\mathcal{A}}$ into a utility distance D_G for G so that $D_G(s, t) = 0$ if and only if for every $\mathcal{A}$ component x in s, there exists an $\mathcal{A}$ component y in t such that $D_A(x, y) = 0$. The utility constraint on the edit function M over G then requires that for any observation $o \in \Theta(R)$, the state s of G reached by o and the state t of G reached by the obfuscated observation $M(o)$ satisfy $D_G(s, t) = 0$.

Now we construct an edit function M by applying EdiSyn to solve Problem 1 for G, $\mathcal{M}$. The resulting edit function is encoded by an obfuscator automaton, similar to a string transducer, containing 440 states and 11,290 transitions. Therefore it is not possible to include the full result in this paper. However, we consider again the violating event sequence o defined in equation (30). This is mapped by the edit function to the obfuscated event sequence

$$M(o) = \Big((1, \mathcal{T}, \mathcal{R})\{2\},\ (3, \mathcal{T}, \mathcal{T})\{3\},\ (4, \mathcal{T}, \mathcal{T})\{3\}\Big), \tag{33}$$

which corresponds to the new state-estimate sequence

$$\begin{aligned} X_1' &= \{(1, 2, 1), (1, 3, 1), (1, 4, 1), (1, 5, 1)\} \\ X_2' &= \{(3, 3, 3)\} \\ X_3' &= \{(4, 4, 4)\}. \end{aligned} \tag{34}$$

Since X_1' contains nonsecret states from which the event $M(o)_2 = (3, \mathcal{T}, \mathcal{T})\{3\}$ can occur, and since X_2' and X_3' each contain only nonsecret states, then joint 1-step opacity is not violated by $M(o)$. Additionally, this is a valid edit since each event that may occur from a state in X_2 may also occur from a state in X_2'.

VI. Conclusion

We considered the problem of synthesizing edit functions that enforce the various notions of K-step opacity. By transforming K-step opacity into a language-based notion, we can apply existing synthesis methods for CSO. To the authors' knowledge, synthesis methods for edit functions enforcing K-step opacity have not been proposed before. We demonstrate this approach on a novel contact-tracing system model. We focused on the effectiveness of edit functions that are not known to the eavesdropper, but a similar approach can be used assuming the edit function is public as in [21]. Additionally, the language-based formulation of K-step opacity could be used to study the problem of synthesizing supervisory control to enforce K-step opacity.

References

[1] J. W. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan, "Opacity generalised to transition systems," *International Journal of Information Security*, vol. 7, no. 6, pp. 421–435, Nov. 2008.

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

[3] A. Saboori and C. N. Hadjicostis, "Notions of security and opacity in discrete event systems," in *2007 46th IEEE Conference on Decision and Control*, Dec. 2007, pp. 5056–5061.

[4] ——, "Verification of initial-state opacity in security applications of DES," in *2008 9th International Workshop on Discrete Event Systems*, May 2008, pp. 328–333.

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

[6] X. Yin, M. Zamani, and S. Liu, "On Approximate Opacity of Cyber-Physical Systems," *IEEE Transactions on Automatic Control*, pp. 1–1, 2020.

[7] A. Saboori and C. N. Hadjicostis, "Verification of K-step opacity and analysis of its complexity," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference*, Dec. 2009, pp. 205–210.

[8] Y. Falcone and H. Marchand, "Enforcement and validation (at runtime) of various notions of opacity," *Discrete Event Dynamic Systems*, vol. 25, no. 4, pp. 531–570, Dec. 2015.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] L. de Alfaro, T. A. Henzinger, and O. Kupferman, "Concurrent reachability games," *Theoretical Computer Science*, vol. 386, no. 3, pp. 188–217, 2007.

[15] X. Yin and S. Li, "Synthesis of Dynamic Masks for Infinite-Step Opacity," *IEEE Transactions on Automatic Control*, pp. 1–1, 2019.

[16] 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.

[17] A. Wintenberg, M. Blischke, S. Lafortune, and N. Ozay, "A General Language-Based Framework for Specifying and Verifying Notions of Opacity," *arXiv:2103.10501 [cs]*, Mar. 2021.

[18] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York, NY: Springer, 2008.

[19] 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.

[20] E. Roche and Y. Schabes, Eds., *Finite-State Language Processing*, ser. Language, Speech, and Communication. Cambridge, Mass: MIT Press, 1997.

[21] 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.

[22] 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.

[23] J. Chan, L. Cox, D. Foster, S. Gollakota, E. Horvitz, J. Jaeger, S. Kakade, T. Kohno, J. Langford, J. Larson, P. Sharma, S. Singanamalla, J. Sunshine, and S. Tessaro, "PACT: Privacy-sensitive protocols and mechanisms for mobile contact tracing." *IEEE Data Engineering Bulletin*, vol. 43, no. 2, pp. 15–35, Jul. 2020.

[24] E. M. Redmiles, "User Concerns 8 Tradeoffs in Technology-facilitated COVID-19 Response," *Digital Government: Research and Practice*, vol. 2, no. 1, pp. 6:1–6:12, Nov. 2020.

[25] Y.-C. Wu, K. A. Sankararaman, and S. Lafortune, "Ensuring Privacy in Location-Based Services: An Approach Based on Opacity Enforcement," *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 33–38, 2014.