Demonstration of Indoor Location Privacy Enforcement using Obfuscation ⋆

Rômulo Meira Góes * **Blake C. Rawlings** * **Nicholas Recker** *
Gregory Willett * **Stéphane Lafortune** *

* *Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122, USA (email:{romulo,bcraw,nrecker,willettg,stephane}@umich.edu)*

Abstract: We apply discrete-event-control-theoretic techniques for opacity enforcement by insertion or deletion of output events to the problem of location privacy enforcement in an indoor environment where users are continuously monitored by IoT devices. We design an obfuscator of user trajectories in a grid model with obstacles. The obfuscator must preserve a secret (e.g., visits to secret cells of the grid), while at the same time enforce feasibility and utility constraints for obfuscated trajectories. We implement the obfuscator to map the true location of the user to an obfuscated location, in real time, using services provided by a data server called the Global Data Plane which records sensor readings from IoT devices and publishes them to subscribers. We explain how scalability of obfuscator synthesis (off-line) and instantiation (on-line) is achieved. We demonstrate the approach on a grid with over 1,500 cells modeling the first floor of a university building, where location estimation is achieved using the ALPS Acoustic Location Processing System.

Keywords: discrete event systems, supervisory control, opacity, location privacy, Internet-of-Things

1. INTRODUCTION

Given the proliferation of Internet-of-Things (IoT) devices that are being deployed inside smart buildings, as well as ubiquitous location tracking apps that rely on GPS, the location and movements of a user are increasingly being tracked, both indoors and outdoors, and this information is often presented unaltered to external observers. The benefits of these smart devices therefore come at a cost to privacy. Hence, an important challenge is to protect privacy without sacrificing the benefits provided by smart devices.

This paper demonstrates a mechanism for enforcing a measure of privacy while preserving a measure of utility when a user's trajectory is being tracked, either indoors or outdoors. The technique used relies on obfuscation of the user's actual trajectory in a given environment, in order to hide all visits to a set of pre-specified *secret locations*. The obfuscated trajectory disclosed to external observers must satisfy this secrecy condition at all times. At the same time, obfuscated trajectories should retain some form of *utility*, i.e., they cannot suppress all information. In this work, utility is captured as a maximum distance constraint between actual and obfuscated positions, at every point in time. Finally, obfuscated trajectories must remain *feasible* in the given environment, e.g., they cannot jump, go through walls, or follow impracticable paths.

⋆ This work was supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and in part by the National Science Foundation under grant CNS-1421122.

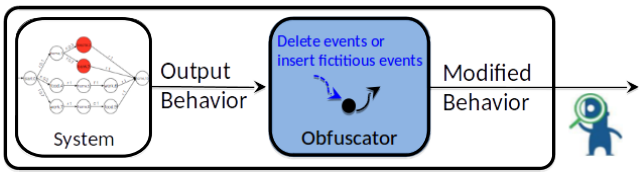

Fig. 1. Obfuscation mechanism using edit functions

Recently, Wu et al. (2018) presented a method for synthesizing so-called *edit functions* that map, in real time, an actual trajectory to a suitably obfuscated one; Fig. 1 illustrates this obfuscation mechanism. Edit functions are a generalized version of the *insertion functions* that are synthesized in Wu and Lafortune (2014) for the enforcement of *opacity*, a general information-flow property that has been studied extensively in the last few years in the context of partially-observed systems (Jacob et al., 2016). Edit functions can delete actual system events and/or insert new fictitious ones (up to a certain bound) when mapping an actual string of events to an obfuscated one.

We adopt the obfuscation framework of Wu et al. (2018) and describe its real-time implementation in the context of a software architecture, written in Python, that first solves the edit function synthesis problem off-line, and then makes suitable edit decisions on-line based on real-time sensor readings of the movements of a user in a pre-specified region modeled using a discrete grid. (In our work, all events are observable, and opacity reduces to to never disclosing visits to secret locations; one can think of the secret locations as the states that violate opacity in the observer of the original partially-observed system.)

In Wu et al. (2018), two symbolic implementations of edit function synthesis are presented and compared. We adopt the second method, based on using the tool SynthSMV (Rawlings, 2016). In this method, edit function synthesis is mapped to a supervisory control problem that is then solved symbolically using SynthSMV, which is based on the symbolic model-checking solver NuSMV (Cimatti et al., 2002). We also employ SynthSMV to compute the actual edit decisions on-line in real time. A visualization tool is included to display actual and obfuscated positions in the grid layout.

We demonstrate how to use this software architecture in an indoor environment and present the results of our deployment of this technique in a university building. This deployment occurred in the context of the Enhanced Conference Room Demonstrator project of the TerraSwarm Research Center (TerraSwarm, 2017). Specifically, we used two features supported by this demonstrator project: (i) real-time indoor tracking of the location of a user using the Acoustic Location Processing System (ALPS) of Rowe et al. at Carnegie Mellon University, USA (WiSE Lab, 2017; Lazik and Rowe, 2012), which was deployed in portions of the first floor of the Clark Kerr Conference Center on the campus of the University of California at Berkeley, USA; and (ii) a secure data server called the Global Data Plane (GDP), which provides services to IoT devices and apps on smart phones or other devices for real-time data logging, subscriptions to logs, archival services, etc. The GDP is being developed by Kubiatowicz et al. at the University of California at Berkeley, USA (Swarm Lab, 2017; Zhang et al., 2015) and its goal is to provide a data-centric "glue" for swarm applications. In the demonstration, the first floor of the Clark Kerr Conference Center is modeled using a grid of about 1,500 cells that capture the layout of the floor (walls, doors, obstacles). Our implementation reads data written in real time to the GDP by an ALPS app running on a smart device and results in an obfuscated log that is written in real time to the GDP for disclosure to the outside world and display on the visualization tool.

The key features of this demonstration include: (i) the ability to synthesize an obfuscator (off-line) for a grid large enough to model a floor of a building consisting of several rooms; (ii) the ability to make edit decisions (on–line) based on the synthesized obfuscator, which is stored symbolically using binary decision diagrams (BDDs); and (iii) the ability to leverage an indoor localization system (ALPS) and a data-centric server for swarm applications (GDP) to implement obfuscation in real time from actual user movements in the instrumented region.

The remainder of this paper is organized as follows. Section 2 reviews the synthesis technique for the obfuscator. Next, Section 3 presents our software architecture for real-time obfuscation, along with simulation results for real-time obfuscation. The demonstration of our software with live readings from a university building is presented in Section 4. Finally, Section 5 compares our work with related approaches in the literature, and Section 6 concludes the paper.

2. SYNTHESIS OF OBFUSCATOR

2.1 Obfuscation by Edit Functions

We start by describing the control problem that must be solved off-line to calculate a suitable obfuscator for a given system model (which may be a grid, as in our example problem, or a more general model). The obfuscation methodology that is demonstrated in this paper has its theoretical foundations in Wu et al. (2018), where an edit function is added as an output interface between the system and the external observers of its behavior (legitimate or malicious), as depicted in Fig. 1. An edit function, or obfuscator in the context of this paper, is allowed to erase genuine system outputs and/or insert some fictitious ones, thereby producing an obfuscated output stream that is seen by the external observers. The necessary technical results for obfuscator synthesis are presented in Wu et al. (2018), and will only be summarized here.

The system model is a discrete-event model, either an automaton or a labeled transition system, where the labels on the transitions between the discrete states are the *events*, which correspond to transitions from one grid cell to another in the grid model of this paper. Since the external observers track the system state (cell location in the grid) based on the sequence of events emitted by the edit function, the role of the edit function is to perform insertions of fictitious events or deletions of actual events (i.e., moves from one grid cell to another) in such a manner that the obfuscated trajectory never visits any of the *secret states* (some grid cells that are pre-specified). However, the obfuscated trajectory must still be a *feasible* one in the system model; it cannot go through walls for example or jump over grid cells. The result of obfuscation is that the true system state will be different from the observed system state (i.e., the "fake" or obfuscated one). Practical considerations might dictate that the difference between the obfuscated state and the true one should be within some bound, under some quantitative measure. In our problem, we quantify such a condition by the number of (valid) moves between the true cell (where the agent is located) and the obfuscated one (as estimated by the external observers).

Synthesizing an edit function that satisfies all of the above conditions is a "reactive synthesis problem with a plant model", as described in Ehlers et al. (2017) for instance. In Wu et al. (2018), this problem is solved by formulating a game between the edit function and the system (acting as "environment"). One must find, if it exists, a winning strategy for the edit function that works under all system behaviors (worst case analysis). The construction of that game and the identification of a winning strategy are described in detail in Wu et al. (2018). In the next section, we focus on the implementation aspects of this synthesis problem as a *supervisory control problem* (Ramadge and Wonham (1987); see, e.g., Ehlers et al. (2017) for an introduction), which are relevant to the present demonstration paper.

2.2 Symbolic Implementation of Obfuscation

As was mentioned in Section 1, we selected the symbolic implementation of obfuscation based on SynthSMV, one of

the two symbolic implementations analyzed in Wu et al. (2018). Its scalability properties were the primary reason for this selection. In short, the system is modeled as a deterministic finite labeled transition system, additional variables are added to the system model to represent the obfuscation decisions, and the desired behavior is formulated as a computation tree logic (CTL) specification; the resulting supervisory control problem has a unique solution that yields the obfuscator. We now discuss this implementation in more detail.

The problem of synthesizing an edit function that simultaneously satisfies the three conditions of secrecy, feasibility, and utility, is formulated as a supervisory control problem, where system outputs are *uncontrollable* events of the "plant", and edit function actions (here, insertions or deletions) are the *controllable* events that are to be enabled or disabled by the "supervisor". In this manner, the solution of the supervisory control problem suitably emulates the game-like structure of Wu et al. (2018) [1], which embeds all valid edit actions when the edit function "plays against" the system, which in turn acts as environment or adversary.

More specifically, in our formulation, the plant model for the supervisory control problem captures all allowed transitions in the system dynamics. We need to track two state variables over these dynamics: the actual state of the user and its obfuscated state. An additional Boolean variable keeps track of whose turn it is to move: the user or the obfuscator. The utility constraint is assumed to be in the form of a maximum distance or budget (in L_1-norm in the case of a grid model), denoted by W, between the actual and obfuscated states. For a solution to exist, this budget may need to be adjusted depending on the system model.

As described in Wu et al. (2018) (cf. Section 5.2), the specification for supervisor synthesis is a CTL formula of the form

$$\mathsf{AG}(\phi_1 \wedge \phi_2 \wedge \mathsf{EF}(\phi_3))$$

where ϕ_1 captures the secrecy constraint (the obfuscated state should never be a secret location), ϕ_2 captures the utility constraint (the budget must always be satisfied), and ϕ_3 captures the ability of the user to make a move in the game. The first two terms are *invariance* constraints, and do not include any additional temporal operators. The latter *reachability* constraint $\mathsf{EF}(\phi_3)$ ensures that the obfuscator never encounters a situation where it has no valid move, which would result in stopping the game and preventing the user from ever moving again. Finally, in our implementation, the number of moves reported by the obfuscator between any two user moves is bounded by variable K, typically chosen to be from 2 to 5 in our experiments.

As argued in Ehlers et al. (2017), for a deterministic plant model (as we have here) and for a specification of the form $\mathsf{AG}(\mathsf{EF}(p))$, there exits a unique maximally-permissive state-based supervisor; this extends naturally to specifications with additional invariance requirements, as in our case. The supervisor is a function that maps each state to the set of valid insertions (which could contain the empty string, corresponding to a deletion) that may follow the previous move of the user and precede its next move. In our demonstrations, we randomly selected *one* insertion from that set whenever the obfuscator played.

[1] This game-like structure was inspired by that of Wu and Lafortune (2014) for the case of insertion actions only.

Figure 2 shows the typical steps involved in using SynthSMV. The system is modeled using the SMV modeling language (specifically, the same input language as NuSMV version 2.6.0 is supported), along with the definition of which events are controllable and uncontrollable, and the specification is written in CTL. Controllability of inputs and synthesis specifications are extensions to NuSMV's modeling language. SynthSMV can return either the supervisor itself or the set of states that satisfy the specification when that supervisor is applied, each either as an explicit set or as an expression in SMV syntax. Figure 2 also includes an example with four states ($Q = \{1, 2, 3, 4\}$), two events ($\Sigma = \{\alpha, \beta\}$), one of which is controllable ($\Sigma_c = \{\alpha\}$), and a reachability specification. SynthSMV can also be called with a previously-computed supervisor and a given state to produce the set events allowed by the supervisor in that state. This can also be combined with NuSMV's interactive mode to simulate a system without having to repeatedly rebuild the symbolic representation, thereby allowing the supervisor to be applied to control a live system.

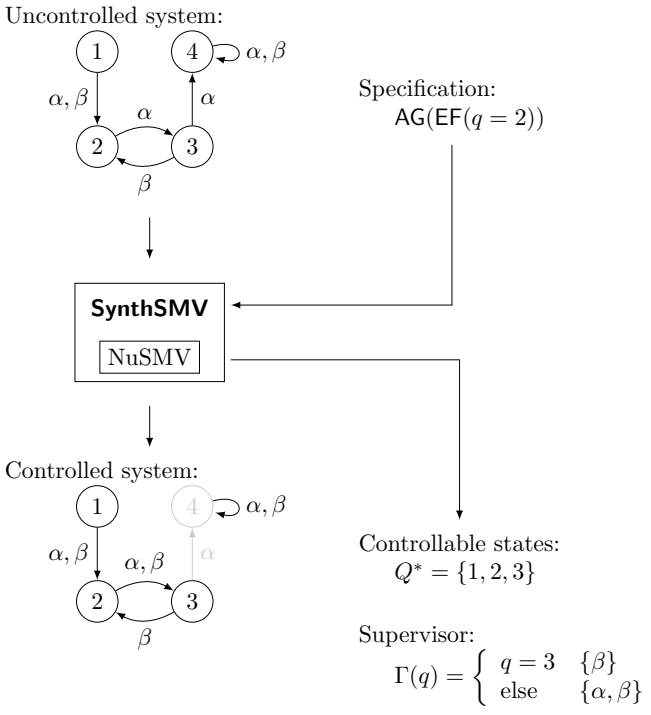

Fig. 2. Interaction with SynthSMV

3. REAL-TIME IMPLEMENTATION OF OBFUSCATOR

In the previous section, the general procedure to synthesize an obfuscator (when one exists) was presented. We now present a software architecture, written in Python, that synthesizes an obfuscator and uses it to make obfuscation decisions in response to live readings. We restrict our attention to agents moving in a grid world, but the

framework could easily be extended to other obfuscation scenarios.

In order to implement real-time obfuscation, we employ two key tools. First, we adopt a data-centric system for IoT applications, the GDP. The GDP infrastructure was presented in Zhang et al. (2015), and it fits the requirements of IoT applications in a natural way. By using such infrastructure, we are able to perform real-time obfuscation, disclosing obfuscated positions to third party applications (e.g., Location-Based Services). In addition to the GDP, we leverage SynthSMV not only to compute the obfuscator, but also as a look-up table using the pre-computed obfuscator. The software architecture is shown in Fig. 3, where the obfuscation application reads/writes information from/to the GDP, and makes queries to SynthSMV. We explain below each of the components and its properties.

3.1 Global Data Plane (GDP)

The GDP is an infrastructure developed to address problems due to the increase of IoT applications. In Zhang et al. (2015), the authors described the disadvantages of connecting smart devices directly to the cloud. To overcome such disadvantages, a data-centric abstraction layer is inserted between IoT applications and the underlying existing computing platforms, e.g., Cloud, Fog, and Gateways (Zhang et al., 2015). Such abstraction is focused around the transportation, the conservation, and the protection of the information.

The foundation of the GDP is the idea of a secure, single-writer log. Each log has a single authorized writer, which is enforced by a writer authentication key. The key also provides authenticity and integrity to the log. In an IoT application, each sensor or computational element has its unique log in the GDP. Moreover, the GDP supports multiple simultaneous readers, each of which must have the appropriate decryption key to decrypt the data. In conclusion, the GDP provides coherent, scalable, and robust communication between logs and applications. For more details of the usage of the GDP, we refer the reader to Zhang et al. (2015) and Swarm Lab (2017).

In our software infrastructure, we read from and write to the GDP. Each agent has a position log in the GDP, and only our application has the key to read it, so that the exact position of each agent is private information. Our application then writes a public log to the GDP to publish the obfuscated position of each agent.

3.2 SynthSMV

SynthSMV can compute an obfuscator to solve the problem described in Section 2, when one exists. In this section, we show how we combine off-line and on-line use of SynthSMV to implement real-time obfuscation efficiently.

In relation to the off-line case, our software uses SynthSMV exactly as previously explained in Section 2. Starting with the actual agent's mobility model, it constructs an obfuscator that controls the "fake" agent's movements. This obfuscator is stored to later be used by our software on-line so that we can avoid repeating the costly synthesis computation.

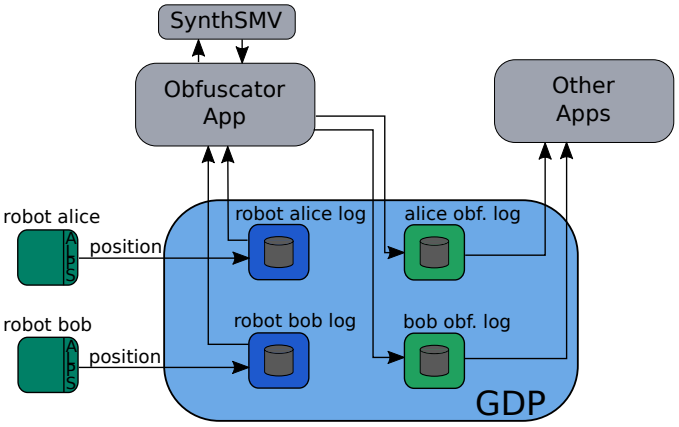

Fig. 3. Real-time implementation of obfuscator using the Global Data Plane (GDP)

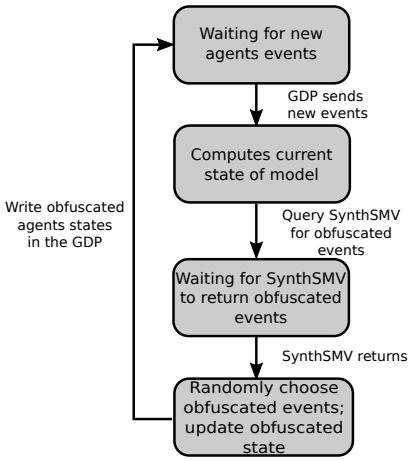

Fig. 4. Software architecture for entire tool incorporating Python core, SynthSMV, reading/writing to/from GDP, and visualization

The on-line usage of SynthSMV is implemented as a sequence of queries made by our software to SynthSMV. Before any queries are made, the previously-computed obfuscator is loaded into SynthSMV. Then, each query simply asks which events are enabled by the obfuscator in a particular state. SynthSMV returns the set of all allowed obfuscation moves from that state of the model. The advantage of using SynthSMV on-line as a look-up table is its efficient BDD-based symbolic representation of the obfuscator. Even when the obfuscator is complex and has a large state space, SynthSMV can respond to the queries within one second, which it is fast enough for real-time applications.

3.3 Obfuscation Application

The obfuscation application is created to manage the data received from the GDP and the queries to SynthSMV. It performs a sequential routine as shown in Fig. 4. We are leaving out the off-line obfuscator computation step, which we assume was done prior to the sequential steps shown in Fig. 4.

The first step, the one that triggers all the other steps, is receiving new position information for the agents from the GDP. Once all agents have updated their respec-

tive current positions, the application can then begin its computation. The second step is to update the internal representation of the agents' positions in our model. Given the updated state, our application queries SynthSMV to receive the set of every move that can be reported without violating the specification.

At this point, we are in the third step of Fig. 4. As soon as SynthSMV returns the allowed obfuscated agents' movements, the application randomly selects one of them. The obfuscated agents' positions are updated according to the selected move, and written to their respective public logs. After writing to the GDP, the obfuscation application returns to its first step where it once again awaits the updates to the agents' positions.

The obfuscation application also includes a visualization tool. The main goal to embed a visual component in our application is mainly to demonstrate the difference between trajectories generated by an agent and its obfuscated trajectory. Figures 5 and 7 are examples of the visual component.

3.4 Simulation Results

To show the applicability of our software architecture, we generated simulated trajectories of agents moving on the first floor of a university building. We generated different scenarios by modifying the size of the grid, the secret location, the budget W, and the bound K.

Figure 5 shows one agent moving in the grid world, along with its obfuscated trajectory. The bold black lines represent walls which the agent cannot pass through. The secret location is represented by the red grid locations. Moreover, we considered the budget value $W = 3$ and the bound value $K = 3$. For this scenario, the off-line SynthSMV query took 5 hours and 12 minutes on a server with an Intel Xeon CPU E5-1650 (3.50GHz) and 64GB of memory running Linux. The large computation time is due to a combination of the large size of the grid, close to 1500 cells, with the walls obstacles.

In order to simulate real-time obfuscation, the agent's positions are updated periodically in the GDP. The trajectories in Fig. 5 represent both the real (dark blue) and obfuscated (light blue) agent's positions. In the building, there are two doors that lead to the patio area. In the scenario of Fig. 5, one of these doors is secret. Therefore, any time the agent goes through the secret door, the obfuscated trajectory has to pass through the other one, as seen in Fig. 5.

4. DEMONSTRATION IN INDOOR ENVIRONMENT

The architecture presented in Fig. 3 assumes that the agents' positions are continually written in their respective logs. However, we did not assume any specific system, and instead left the architecture open to any type of agent localization method (e.g., GPS, video input, etc.). We now present a specific demonstration of indoor real-time obfuscation. First, we discuss an indoor localization methodology used as input to our software and then we present the results of real-time obfuscation for a specific scenario for the first floor of a university building.

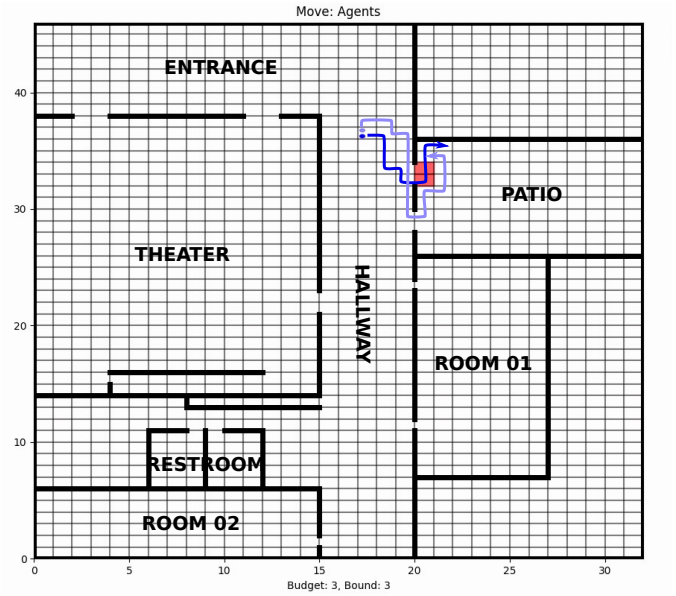

Fig. 5. Grid mapping of the first floor of the Clark Kerr Conference Center at UC Berkeley; the secret location (in red) is one of the doors; the trajectories of the real (obfuscated) agent are shown in dark (light) blue

4.1 ALPS — Acoustic Location Processing System

The Acoustic Location Processing System (ALPS) was developed by Lazik and Rowe (2012). It uses a communication scheme in the ultrasonic audio bandwidth to provide sub-meter accurate indoor localization of mobile devices such as smartphones and tablets (Lazik and Rowe, 2012; WiSE Lab, 2017). The communication scheme is imperceptible to humans, since it exploits a bandwidth space above the human hearing frequency spectrum, yet perceptive by the mobile devices.

The tracking provided by ALPS uses off-the-shelf audio speakers placed in strategic positions to localize the mobile devices. Therefore, the area in which tracking will be performed needs to first be instrumented. Figure 6a shows an ALPS beacon placed in the university building from our demonstration, and Fig. 6b shows the hallway that was instrumented with ALPS beacons in our obfuscation scenario. Four beacons were placed in the hallway, one at each corner; this allowed accurate position information within the hallway to be published to the GDP every second using a tablet device.

4.2 Demonstration

The instrumented section of the hallway, shown in Fig. 6b, corresponds to grid positions $15 \leq x \leq 20$ and $15 \leq y \leq 25$ in Fig. 5. Given that the agent can only be tracked in the hallway, we simplify the previous grid model of Fig. 5. The model still considers one agent moving in the grid world, but the secret location and the grid layout are now defined as in Fig. 7. The coordinates in Fig. 7 matched the coordinates instrumented by ALPS, where the instrumented hallway is 4 meters wide and 25 meters long (we partially show the hallway grid).

The agent walks freely in the hallway, constrained by a table positioned in the middle of the corridor. The table is

(a) ALPS device — Photo courtesy of Niranjini Rajagopal and Anthony Rowe

(b) ALPS-instrumented hallway on the first floor of the Clark Kerr Conference Center

Fig. 6. Photographs the hardware and building used in the live demonstration

modeled as obstacle locations (in gray) in Fig. 7. Based on the location of the table, we defined as the secret location the entire space left to the table (in red), meaning that an outsider observer should never know when the agent walks on the left side of the table (e.g., the agent does not want others to know when it is eating).

With the model defined, we set the free parameters W and K such that there exists an obfuscator that satisfies the defined constraints. In this example, we selected $W = 3$ and $K = 3$ to obtain a solution to the obfuscation problem.

Next, using the architecture of Fig. 3, a single agent moved freely in the instrumented hallway. The agent's position was recorded by ALPS and published to the GDP. Simultaneously, the Obfuscator App created an obfuscated trajectory based on the agent's actual trajectory, as explained in the previous section.

We present one of agent's trajectories in Fig. 7. While the agent (dark blue) walks within the secret region, its obfuscated trajectory (light blue) indicates that the agent never enters the secret region. The numbers in the trajectory reflect the agent's movements and the respective obfuscated movements. Therefore, there exist repeated numbers in the obfuscated trajectory given that the obfuscator could move up to three times ($K = 3$) before the next agent update.

It is interesting to compare the obfuscated actions numbered by 8 with those numbered by 5. We see that the obfuscated actions 8 lead the obfuscator to the maximum of its allowed movements ($K = 3$). It does so that it prepares itself for an unknown future of the agent. After move 8, the agent is located in the boundary between secret and non-secret locations. Therefore, move 8 prepares the obfuscated trajectory for any possible future movement of the agent. In contrary, movement 5 does not need to deal with such a situation. Given that the agent will remain in the secret location for any action it chooses after move 5, the obfuscator needs only one action to prepare itself for the future actions of the agent.

We remark that by using SynthSMV, the obfuscated movements are calculated within a second, before the agent updates its position. Furthermore, the trajectory in

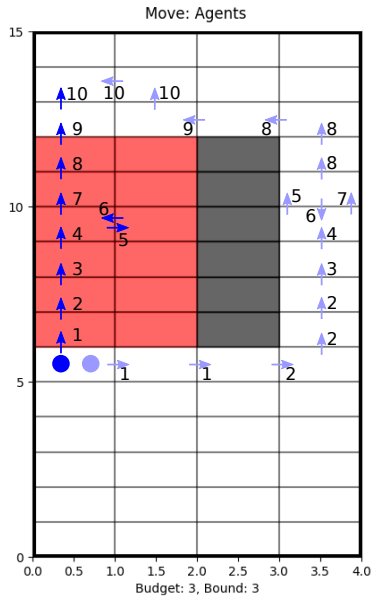

Fig. 7. Actual and obfuscated trajectories overlaid on the grid model, using real-time sensor measurements

dark blue is created as a private log in the GDP, and it is only accessible by the Obfuscator App. However, the obfuscated one is published as a public log.

5. RELATED WORK

The objective of this paper was to demonstrate real-time obfuscation policies that ensure privacy and utility. Obfuscation techniques for location privacy were introduced by Duckham and Kulik (2005a,b). The idea of obfuscation is to degrade the information of a person's location in order to obtain privacy. In the work of Duckham and Kulik (2005a), the information is degraded in an imprecise manner such that the privacy of the individuals is maintained. However, the study is only made for single queries and a dynamic query model is not studied. The work of Ardagna et al. (2011) strengthened the prior work by providing quantitative measurements about the quality of the obfuscation. A downside of these techniques is that they perform probabilistic operations, which could provide inconsistencies when analyzing a dynamical scenario (e.g., agents moving through obstacles). Our model-based approach guarantees that such inconsistencies will not occur.

Another related research area is that of differential privacy. Indeed, the attacker model and the problem formulation presented here are inspired by work on differential privacy. In the work by Dwork (2006), the differentially-private mechanism does not specify any utility guarantees. The first work to include utility guarantees in differentially-private mechanisms was the work of Ghosh et al. (2012). In this context, our obfuscator can be viewed as a discrete logic counterpart of differentially-private mechanism with utility constraints.

The work of O'Kane and Shell (2015) is closely related to ours. They also investigated privacy and utility requirements in a model-based manner. However, they specified theses constraints as pairwise requirements on states, where privacy is transformed to state indistinguishabil-

ity, and utility to state distinguishability. The solution is given via graph coloring. In our work, the constraints are described as temporal logic specifications. Moreover, the solution with the obfuscation mechanism is more general than the one provided by O'Kane and Shell (2015), as it allows modification of the system's output events.

6. CONCLUSION

We have described how to perform obfuscation, a form of opacity enforcement based on editing the output behavior of a system, in real time for systems with large state spaces. This includes the off-line synthesis of a suitable obfuscator and its on-line implementation. We have demonstrated our approach in the context of location privacy in an indoor environment where the user's location is tracked continuously by an acoustic positioning system and where an obfuscated position that hides visits to secret locations must be disclosed in real time to the outside world via a data server. Overall, we believe this demonstration shows the practicality of achieving indoor location privacy using the opacity enforcement mechanism of edit functions developed for discrete event systems.

ACKNOWLEDGEMENTS

It is a pleasure to acknowledge the work of Niranjini Rajagopal and Patrick Lazik in setting up the ALPS system in the Clark Kerr Conference Center, and the help of Nitesh Mor in working with the GDP. We also thank all the members of the Enhanced Conference Room Demonstrator project in the TerraSwarm Research Center.

REFERENCES

Ardagna, C.A., Cremonini, M., di Vimercati, S.D.C., and Samarati, P. (2011). An obfuscation-based approach for protecting location privacy. *IEEE Transactions on Dependable and Secure Computing*, 8(1), 13–27.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV 2: An OpenSource tool for symbolic model checking. In *Computer Aided Verification*, Lecture Notes in Computer Science, 359–364. doi:10.1007/3-540-45657-0_29.

Duckham, M. and Kulik, L. (2005a). *A Formal Model of Obfuscation and Negotiation for Location Privacy*, 152–170. Springer Berlin Heidelberg, Berlin, Heidelberg.

Duckham, M. and Kulik, L. (2005b). *Simulation of Obfuscation and Negotiation for Location Privacy*, 31–48. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dwork, C. (2006). Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ICALP'06, 1–12. Springer-Verlag, Berlin, Heidelberg.

Ehlers, R., Lafortune, S., Tripakis, S., and Vardi, M. (2017). Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems: Theory and Applications*, 27(2), 209–260.

Ghosh, A., Roughgarden, T., and Sundararajan, M. (2012). Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6), 1673–1693.

Jacob, R., Lesage, J., and Faure, J. (2016). Overview of discrete event systems opacity: Models, validation, and quantification. *Annual Reviews in Control*, 41, 135–146.

Lazik, P. and Rowe, A. (2012). Indoor pseudo-ranging of mobile devices using ultrasonic chirps. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, 99–112. ACM, New York, NY, USA.

O'Kane, J.M. and Shell, D.A. (2015). Automatic design of discreet discrete filters. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 353–360.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes. *SIAM J. Control & Optimization*, 25(1), 206–230.

Rawlings, B.C. (2016). *Discrete Dynamics in Chemical Process Control and Automation*. Ph.D. thesis, Carnegie Mellon University. URL `http://repository.cmu.edu/dissertations/862/`.

Swarm Lab (2017). Global Data Plane. `https://swarmlab.eecs.berkeley.edu/projects/4814/global-data-plane`. Accessed: 2017-11-06.

TerraSwarm (2017). TerraSwarm Research Center. `https://www.terraswarm.org`. Accessed: 2017-11-06.

WiSE Lab (2017). Acoustic Location Processing System. `http://wise.ece.cmu.edu/redmine/projects/alps/wiki`. Accessed: 2017-11-06.

Wu, Y.C. and Lafortune, S. (2014). Synthesis of insertion functions for enforcement of opacity security properties. *Automatica*, 50(5), 1336 – 1348.

Wu, Y.C., Raman, V., Rawlings, B.C., Lafortune, S., and Seshia, S.A. (2018). Synthesis of obfuscation policies to ensure privacy and utility. *Journal of Automated Reasoning*, 60(1), 107–131.

Zhang, B., Mor, N., Kolb, J., Chan, D.S., Goyal, N., Lutz, K., Allman, E., Wawrzynek, J., Lee, E., and Kubiatowicz, J. (2015). The cloud is not enough: Saving IoT from the cloud. In *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'15. USENIX Association, Berkeley, CA, USA.